HotMouse

- musstyrning via diskreta tangenttryckningar

Examensarbete i datavetenskap, 20p

av Ola Samnegård

2002

Sammanfattning

Personer med rörelsehinder kan ha svårt att använda en datormus. Detta examensarbete har gått ut på att utveckla metoder för att styra muspekaren via diskreta tangenttryckningar. Dessa har, så långt som det har varit möjligt, implementerats för att fungera i hela Windows 98 och Windows Millennium.

Fyra olika metoder har utvecklats. De kan enklast beskrivas som stegvis förflyttning med variabel steglängd, rekursiv skärmuppdelning, hopp mellan intressanta ställen och numrering av intressanta ställen. De tre förstnämnda av dessa har jag lyckats skapa någon form av fungerande implementation av. Hos ett par av metoderna har även effektiviteten analyserats matematiskt.

Implementationerna har bildat programmet HotMouse som även har testats av några personer med mer eller mindre anknytning till systemutveckling eller funktionshinder.

Abstract

There can be difficulties for people with physical disabilities to use a computermouse. The aims of this Master Thesis is to develop methods to move the cursor through discrete key presses. They have, as far as possible, been implemented to work in Windows 98 and Windows Millennium.

Four different methods have been developed. They can be described as relative movement by various degrees, recursive division of the screen, jumps between interesting places and numbering of interesting places. For the three methods first mentioned I have succeeded creating some working implementations. Concerning a couple of the methods I also analyzed the efficiency mathematically.

The implementations has formed the programme HotMouse that has been tested by some persons more or less connected to system development or physical disability.

Förord

Liksom ingen på denna jord kan prestera något på egen hand, så har också jag fått otroligt mycket hjälp att nå fram till denna examen. Om jag skulle nämna alla som har varit till hjälp på vägen, så skulle listans omfattning blivit ohållbart lång. Jag vill tacka alla dem som på något sätt har bidragit till att jag är där jag är idag. Några få av alla dessa vill jag ändå nämna.

Vare sig man vill erkänna det eller ej, tror jag att den familj man föds in i, har ett stort ansvar för den personliga utvecklingen. Då man som jag har ett funktionshinder är dess roll ännu viktigare. Jag vill därför tacka min familj och släkt, som på alla sätt har varit en motivations- och inspirationskälla till att utforska världen, alltsedan späd ålder.

Jag vill också rikta en rad speciella tack till personer som har varit en stor hjälp under examensarbetets gång. Ett stort tack skänkes till mina två handledare, Paul Reinerfelt som är på datavetenskap och Håkan Eftring på Certec. Tack också till Per-Olof Hedvall ute på Furuboda. Jag vill dessutom tacka Björn Breidegard, samt alla andra på Certec som har varit till en stor hjälp.

Kanske blir detta examensarbetet startpunkten för mig för att utveckla program för rörelsehindade. Det är i varje fall min förhoppning. Jag har nämligen upptäckt att det inte har satsats så mycket på att utveckla mjukvara till denna grupp. I och med att jag själv har ett rörelsehinder, så har jag både en särskild kompetens och en stor motivation till denna typ av programutveckling. Men hur det blir med den saken får framtiden utvisa. En sak vet jag iallafall. Jag vill använda mitt kunnande till att gynna människors sanna väl.

Innehållsförteckning

1 Inledning

1.1 Idéns härkomst

Idén bakom detta examensarbete kom från början från ett önskemål jag själv haft i många år. Jag har själv en CP-skada sedan födelsen, som bland annat gör att jag inte kan använda mus. Tangentbord kan jag emellertid hantera med hjälp av en hjälm med en pinne på huvudet.

Då jag fick mitt första ritprogram (se kap 2.5), upptäckte jag att det fanns program där man på ett enkelt sätt kunde styra markören via tangentbordet. Jag vände mig efter en tid till de datatekniker jag hade kontakt med för att höra om de kunde göra något program eller någon hårdvara som gjorde att ett liknande styrsätt fungerade i hela Windows. De gjorde sina försök, men gav upp på grund av tidsbrist. När det sedan blev dags för mig att göra ett 20 poängs examensarbete i datavetenskap bestämde jag mig för att själv göra detta.

1.2 Certec

Då jag hade valt detta område, föll det sig naturligt att söka samarbete med Certec (Centrum för rehabiliteringsteknik) som är en avdelning inom Institutionen för design-vetenskaper på LTH. De nappade på idén och arbetet har därför gjorts där under Håkan Eftrings handledning.

Certecs målsättning med sin forskning och utbildning är att människor med funktions-nedsättningar skall få bättre förutsättningar genom en mer användarvärd teknik[1], nya designkoncept och nya individnära former för lärande och sökande. Arbetet börjar i människan och slutar i människan samtidigt som både process och resultat ofta har en genuint teknisk karaktär.

2 Bakgrund

2.1 Mushantering – ett problem

Personer med olika former av rörelsehinder kan ha svårt att använda en vanlig dator-mus. Detta kan ha flera orsaker. En del kan ha rumsmässiga begränsningar i sin rörelse-förmåga. Andra kan ha ofrivilliga rörelser eller skakningar som försvårar tids- och rumsmässig precision. Det kan även finnas andra orsaker till begränsad precisions-förmåga.

Det är också relativt vanligt med andra funktionshinder i kombination med rörelse-hinder. Det kan röra sig om syn- eller hörselskador, kognitiva funktionshinder, etc. Varje funktionshinder, och i viss utsträckning varje person med funktionshinder, behöver sina egna datoranpassningar för att datoranvändandet skall gå så smidigt som möjligt.

[1] användarvärd teknik = teknik som personer anser vara värd att använda. Se www.certec.lth.se/manus/

2.2 Samhällets insatser

Vart i samhället vänder man sig som rörelsehindrad för att få hjälp med datoranpassningar? Svaret är att man i första hand skall vända sig till sin lokala hjälpmedelscentral. Men dessa har inte tillräckligt stort upptagningsområde för att få rutin på att hitta den bästa lösningen för varje person.

Det finns även åtta REDAH-center (REDAH = REgionala center för DAtorbaserade Hjälpmedel) i landet. Dit får man i princip bara komma om den lokala hjälpmedelscentralen på orten misslyckas helt med att hitta något styrsätt till datorn överhuvudtaget. Dessa centra blir därigenom fokuserade på att hjälpa de personer som har allra svårast funktionshinder. Det rör sig t.ex. om personer med mycket grava rörelsehinder som bara kan styra datorn med hjälp av en eller ett par kontakter.

2.3 Avgränsning av målgrupp

Som antytts ovan är rörelsehindrade personer som klarar sitt datoranvändande hjälpligt inte någon högprioriterad grupp att effektivisera datoranvändningen för. Just därför vill jag hålla mig inom denna grupp. Jag har valt att koncentrera mig på personer som trots sitt rörelsehinder kan hantera någon form av tangentbord, men har problem med att hitta något effektivt sätt att styra muspekaren.

Även i denna grupp kan det ofta förekomma personer som har andra funktionshinder utöver sitt rörelsehinder. För att få en renodlad problemställning kom detta examensarbete att koncentreras på att hitta lösningar för personer med enbart rörelsehinder.

2.4 Problem med befintliga styrsätt

2.4.1 Snabb reaktionsförmåga

Man kan tycka att det redan idag finns gott om alternativ till en vanlig datormus. De allra flesta av dessa hör till endera av tre grupper:

- Positionsstyrning
- Styrning av riktning och hastighet
- Avsökande av skärmen.

Positionsstyrning styr, likt en vanlig datormus, muspekaren direkt. Några exempel på detta är huvudmöss[2], som känner av huvudrörelser, och rullbollar av olika storlekar[3] (se bild 2.1)

Bild 2.1 Från vänster: HeadMouse-låda, rullkulan Tumbelina och rullbollen Expert Mouse

[2] Till exempel HeadMouse på sidan http://www.headmouse.com/access/headmouse/
[3] Rullkulan Tumbelina finns på http://www.handy.no/hardware/mus/thumbelina.htm och rullbollen Kensington Expert Mouse http://www.handy.no/tastaturer/Kensington.htm

Det finns de som istället *styr riktning och i vissa fall hastighet* på muspekaren genom att man håller något på ett fixt ställe under den tid man önskar att musrörelsen pågår. Till denna grupp hör en mängd varianter av styrspakar[4], men även Windows inbyggda system för att styra muspekaren via det numeriska tangentbordet (se bild 2.2).

Bild 2.2 Joysticksmus och numeriskt tangentbord

Slutligen finns det olika *avsökande system*[5], som automatiskt först avsöker skärmen med en linje och sedan avsöker linjen med en punkt. Linjen kan t.ex. vara vågrät och gå över skärmen uppifrån och ner eller rotera kring en utgångspunkt (se bild 2.3). Användaren stoppar avsökandet genom att ge något kommando då önskad linje eller punkt är markerad.

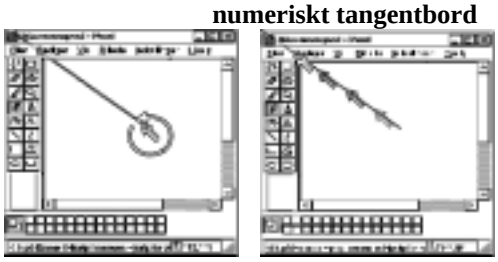


Bild 2.3 IRIS – Radarmus

Dessa tre grupper har gemensamt att de kräver en hög precision hos användaren. Det vill säga att man kan göra något på en tid som är mycket kortare än tiden musrörelsen pågår. I positionsstyrning gäller det att avbryta rörelsen. I styrning av riktning handlar det om att släppa det styrdon som man håller i ett fixt ställe. Och i avsökande av skärmen skall man ge ett kommando.

Generellt kan man säga att reaktionsandelen av tiden för musrörelsen blir det relativa felet, även på utslaget. Antag till exempel att en person har reaktionstiden 0,2 s och han vill flytta musmarkören 20 cm med en felmarginal (stoppsträcka) på max 0,4 cm. Det tillåtna relativa felet är alltså 0,4/20=0,02=2%. Då krävs det en förflyttningstid på minst 0,2/0,02=10s. Detta får till följd att dessa styrsätt blir mycket långsamma för personer vars rörelsehinder medför lång reaktionstid.

I Windows inbyggda system för att styra musmarkören via tangentbordet används det numeriska tangentbordet för att flytta musmarkören i åtta riktningar. Här går de diagonala riktningarna $\sqrt{2}$ gånger så fort som de våg- och lodräta. Detta ger att om man vill kunna stanna med samma felmarginal i alla riktningar behöver pekarhastigheten i horisontal- och vertikalled minskas med $\sqrt{2}$. Då kan en sådan förflyttning ta 14 sekunder.

I avsökande system delas rörelsen in i två som på något sätt är ortogonala. Detta gör också att tiden förlängs.

2.4.2 Många tangenttryckningar

För att komma ifrån kravet på snabb reaktionsförmåga hos användaren bör musstyrningen, likt det normala tangentbordsanvändandet, bygga på diskreta användarkommandon. Det vill säga, de kommandon användaren ger skall inte vara tidskritiska, vare sig längden på kommandot eller tiden mellan två kommandon.

[4] Olinder & Westerberg AB:s Joysticksmus finns på http://www.olinder-westerberg.se/dokhtm/jmus.htm
[5] Till exempel IRIS – Radarmus på sidan http://www.auxilior.com/dk/default.php3?location=radar

Ett sådant alternativ i de flesta Windowsprogram idag är att tabulera sig fram mellan olika kontroller. Det är emellertid också tidsödande då man måste passera kontrollerna enligt en fix sekvens och personer med rörelsehinder ofta har stora begränsningar i antalet tangenttryckningar per minut.

De rena musstyrningar jag hittade som bygger på diskreta användarkommandon är JoySim[6] och rutnätsmodellen i VoiceMouse[7] och Dragon NaturallySpeaking[8].

JoySim flyttar musmarkören stegvis då man trycker på kontakter för de våg- och lodräta riktningarna. I en meny kan man ställa in steglängden som den skall hoppa (se bild 2.4). En stor nackdel med denna produkt är att det är mycket omständligt att byta steglängd. Och om man skall välja en fix steglängd, är man tillbaka till den oönskade konflikten mellan snabbhet och precision.

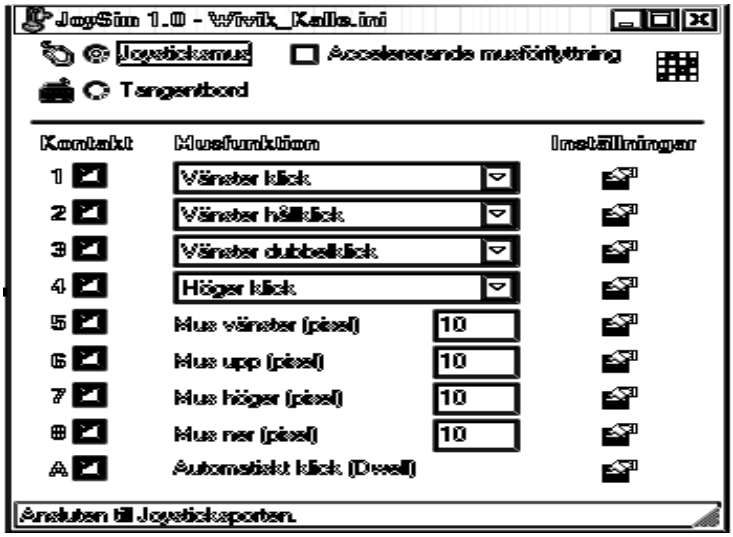
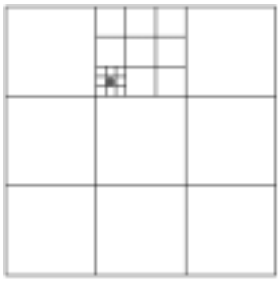


Bild 2.4 Inställningsfönstret i programmet JoySim.

2.4.3 Konsekvent röst



Rutnätsmodellen i VoiceMouse och Dragon NaturallySpeaking delar in skärmen i tre gånger tre rutor (se bild 2.5). Man väljer ruta genom ett röstkommando. Programmen delar sedan in den valda rutan på samma sätt rekursivt tills rutan är så liten att man kan klicka. De här programmen förutsätter en konsekvent röst, vilket inte alla har.

Bild 2.5 Rekursiv skärmindelning i VoiceMouse och Dragon NaturallySpeaking

2.5 Mitt tidigare angränsande arbete

Som framgått tidigare går det oftast redan att ta sig fram via tangentbordet, om än långsamt, i de flesta Windowsprogram. Vad som däremot inte går är att skapa bilder med något av de någorlunda avancerade ritprogrammen utan att använda mus.

Redan som barn försökte jag, trots min CP-skada, skapa bilder på alla sätt jag kunde komma på. Då jag som sexåring fick lära mig att använda skrivmaskin, använde jag genast denna för att skapa bilder. Om jag skrev ett O, ett ö och ett bindestreck på samma plats på pappret, fick jag detta ansikte: ö.

Då jag fick min första dator, en ABC 80, och började lära mig programmera i Basic, skapade jag bilder genom att programmera in vilka ”pixlar” (de var 0,25 cm^2 stora) som

[6] se http://www.anycom.se/products/JoySim/
[7] se http://www.htspeech.se/HTSM.htm
[8] se http://www.lhsl.com/naturallyspeaking/

skulle tändas. 1989 fick jag mitt första ritprogram, Harvard Graphics[9], till en 286:a. Harvard Graphics är egentligen ett program för att skapa presentationer av diagram, men innehåller även en ritfunktion. Denna ritfunktion gick att styra helt utan mus. Markören flyttades stegvis med hjälp av piltangenterna. Det gick att dubblera och halvera steglängden, med plus- respektive minustangenten. Den stora begränsningen var att man inte kunde göra några färgövergångar. I den version jag hade kunde jag dessutom bara använda 16 distinkta färger.

När jag skulle göra specialarbete på gymnasiet, beslutade jag mig därför för att skapa ett eget ritprogram. Detta har jag fortsatt att utveckla utanför specialarbetets ram. I SamneGraphics[10], som detta ritprogram heter, styrs markören precis som i Harvard Graphics. Den stora skillnaden är att man lätt kan göra mjuka färgövergångar och få halvtransparenta objekt.

2.6 Övergripande mål

Med anledning av problembeskrivningarna i detta kapitel är målet med detta examensarbete att utveckla och implementera metoder för att effektivt flytta muspekaren dit man vill med hjälp av tangentbordet. Detta skall ske genom ett minimalt antal diskreta tangenttryckningar, utan att systemet blir för svårhanterligt. Dessa metoder skall fungera i hela Windows, inte bara i ett enskilt windowsprogram.

3 Ursprungliga idéer

Det är lätt att inse att det inte går att hitta något universellt styrsätt som är effektivt i alla situationer. Detta beror på att man använder musen på olika sätt i olika windows-program.

Jag ville därför utveckla skilda metoder för olika sätt att använda muspekaren på, detta för att få de mest effektiva lösningarna för olika personer i olika programmiljöer i Windows.

3.1 Uppdelning av problemet i kategorier

En av huvuduppgifterna i detta examensarbete var att hitta metoder för att genom ett minimalt antal diskreta tangenttryckningar flytta musmarkören till ”intressanta” ställen på skärmen. Med ”intressanta ställen” menas ställen där det händer något specifikt då man klickar där med musen, t ex kontroller, länkar, knappar, fönster.

Men i vissa fall kan man vilja komma åt pixlar på skärmen som inte systemet kan veta är intressanta, till exempel vid förflyttning av objekt och i ritprogram. Detta kräver egna metoder för att båda situationerna skall lösas effektivt.

En annan distinktion som man måste göra är om man önskar en absolut eller relativ förflyttning, det vill säga om man önskar att förflyttningen skall vara beroende av utgångsläget. Det senare kan vara en fördel om man önskar en känsla av rörelse hellre än en pekfunktion, till exempel i ritprogram.

[9] se http://www.softhome.com.tw/soft/harvard1.htm
[10] se http://www.student.lu.se/~dat98osa/konst.html

Vi har alltså denna uppdelning:

	Förflyttning till valfri pixel	Förflyttning till intressanta ställen
Absolut förflyttning	Rutnätsmodellen	
Relativ förflyttning	JoySim	

Tabell 3.1

I tabellen har jag placerat in rutnätsmodellen från VoiceMouse och Dragon NaturallySpeaking samt JoySim på deras respektive rätta plats.

3.2 Konkreta idéer på lösningar

På grund av JoySims begränsningar och på grund av att rutnätsmodellen bara gick att styra med rösten, kom detta examensarbete att försöka ge lösningar till alla de fyra delområdena. Nedan framgår mina idéer till lösningar.

	Förflyttning till valfri pixel	Förflyttning till intressanta ställen
Absolut förflyttning	Rutnät med tangentbord	Numrering av intressanta ställen
Relativ förflyttning	Stegvis förflyttning med variabel steglängd	Hopp till intressanta ställen

Tabell 3.2

3.2.1 Rutnät med tangentbord

Rutnätsmodellen i VoiceMouse och Dragon NaturallySpeaking tillhör, som framgår av tabell 3.1, kategorin ”Absolut förflyttning till valfri pixel”. Detta är effektivt om man vill klicka på vilket ställe som helst på skärmen, oberoende av utgångsläge, men man får ingen känsla för förflyttningen från en punkt till en annan. En av de första idéerna bakom detta examensarbete var att vidareutveckla denna rekursiva skärmuppdelnings-idé och anpassa den till tangentbord.

3.2.2 Stegvis förflyttning med variabel steglängd

En relativ förflyttning till valfri pixel, med hjälp av diskreta användarkommandon, måste bli någon form av stegvis förflyttning. JoySim är ett exempel på detta. Nackdelen med detta program var att man inte enkelt kunde byta steglängd. Min första tanke var att föra över styrsättet från Harvard Graphics och SamneGraphics till hela Windows. Det skulle också gå att skifta steglängd på fler sätt än dubblering och halvering med en tangenttryckning. Detta sätt blir effektivt, när det är verkligt nödvändigt att simulera musrörelser, till exempel i ritprogram. En ytterligare tänkbar förbättring är simulering av mjuka musrörelser med hjälp av bezier-kurvor från numerisk analys.

3.2.3 Numrering av intressanta ställen

Om systemet kan känna till vilka ställen som är intressanta, om dessa inte ligger för tätt och om antalet ställen är större än fem, är sättet som spar flest tangenttryckningar följande:

Numrera de intressanta ställena genom tryckning på en funktionstangent. Sedan är det bara att slå det troligtvis tvåsiffriga numret på det önskade stället. Detta är en form av absolut förflyttning till intressanta ställen. En nackdel med detta sätt kan vara att man efter funktionstangenttryckningen måste titta på skärmen för att se numren. Detta blir en fördröjning för dem som inte samtidigt kan titta på skärmen och trycka ned tangenter. En annan nackdel kan vara att det inte uppskattas att skärmbilden störs.

3.2.4 Hopp till intressanta ställen

Om användaren inte önskar att skärmen fylls av referensnummer, är ett alternativ att man skall kunna hoppa till intressanta ställen med hjälp av det numeriska tangentbordet. Musmarkören hoppar till närmaste intressanta ställe i angiven riktning.

4 Grundförutsättningar för implementation

4.1 Hitta lediga tangenter

Då det var önskvärt att programmet fungerar ihop med så många program som möjligt, fick jag begränsa mig till att använda tangenterna på det numeriska tangentbordet. Deras funktioner i andra program går att komma åt via andra tangenter, vilket gör att de ej är oumbärliga. Dock har andra producenter av hjälpprogram för funktionshindrade utnyttjat samma faktum. Dessa program utnyttjar numeriska tangentbordet då Numlock är av. För att undvika kollisioner valde jag då att bara utnyttja dessa tangenter då Numlock är på.

Dessa tangenter producerar vanligtvis samma tecken som andra, vilket gör att man måste se till koden för själva tangenten för att urskilja den. På grund av att den vanliga enter-tangenten visade sig ha samma tangentkod som enter-tangenten på det numeriska tangentbordet, kunde denna ej användas.

De numeriska tangenter som stod till mitt förfogande var alltså 15 stycken:

[0], [1], [2], …, [9], [/], [*], [-], [+] och [,]

4.2 Välja programspråk

För att kunna koncentrera arbetet på det centrala, ville jag använda en programmerings-miljö som gjorde det enkelt för mig att bygga upp kontaktytor.

Även programmeringsmiljöer är väldigt olika väl anpassade till att användas utan mus. Under hela min utbildning vid universitetet har jag försökt få tag i någon utvecklings-miljö för Java som skulle vara lätthanterlig utan mus. Detta har jag inte lyckats med.

Därför var det naturligt att välja ett språk med en utvecklingsmiljö som jag visste fungerade. Det blev Borland Delphi, som jag har mycket erfarenhet av, bl.a. från utvecklingen av SamneGraphics (se kap. 2.5).

Då alla Windowsprogram blir mer lätthanterliga för mig tack vare resultatet av detta examensarbete, så har jag nu fått en större flexibilitet vad gäller programspråk.

4.3 Lägga programmet i System Tray

För normalanvändaren är det önskvärt att programmet bara fungerar och syns så lite som möjligt. Det bör dock på något sätt visas status för programmet på skärmen. När så önskas bör man lätt kunna få upp ett fönster för att ändra inställningar på programmet. Lösningen på dessa problem är att minimera programmet till System Tray[11].

Detta gjordes med hjälp av en gratis delphi-komponent vid namn TrayIcon från Tempest Software[12]. När denna osynliga komponent finns på programmets huvud-fönster, lägger programmet sig automatiskt i System Tray vid minimering.

Dessutom trodde jag att det var nödvändigt att lägga programmet i System Tray, för att kunna fånga upp tangenttryckningar fast programmet ej var i fokus. Det visade sig dock senare, att så inte var fallet. De tidigare nämnda fördelarna med att lägga programmet i System Tray var ändå tillräckliga skäl till att detta skulle göras.

4.4 Fånga tangenttryckningar

Den allra viktigaste nöten att knäcka i detta examensarbete var hur jag skulle kunna fånga upp vissa tangenttryckningar utan att programmet var i fokus. En lång tid försökte jag att haka på en meddelandehanterare på min SysTray-applikation utan framgång. Det som till slut löste problemet var att jag fick tips om en hemsida[13], där jag hittade den färdiga gratiskomponenten FISHotKey. Denna komponent tar full kontroll över en viss tangenttyckning, oavsett om programmet är i fokus, ligger dolt bakom andra program eller är minimerat på något sätt.

En nackdel med denna komponent var att en sådan komponent bara kunde ta hand om en tangent. Detta gjorde att jag fick skapa 15 sådana komponenter, en för varje tangent som jag ville använda.

4.5 Hitta intressanta ställen på skärmen

För att kunna komma till intressanta ställen på skärmen, måste man på något sätt veta var dessa befinner sig. Detta var inte helt enkelt, eftersom det gäller att få denna information från andra program. Det som löste problemet, i varje fall delvis, var MSAA (Microsoft Active Accessibility[14]).

[11] Den högra delen av aktivitetsfältet, där bl.a. klockan visas
[12] se http://www.tempest-sw.com/
[13] se http://www.torry.net/keysandkeyboard.htm
[14] se http://www.msdn.microsoft.com/library/en-us/msaa/msaastart_9w2t.asp

Detta är en verktygslåda med metoder för att göra program mer lättillgängliga för personer med funktionshinder. Här fann jag metoden *AccessibleObjectFromPoint(p,&pAcc, &varChild)* som given skärmkoordinater returnerar vilket tillgängligt objekt som finns på dessa koordinater. MSAA är i huvudsak gjort för C, C++ och Visual Basic. Därför fick denna metod läggas i en dll, som i sin tur anropas från Delphi.

Då man har tillgång till denna metod skulle man lätt kunna hitta de intressanta ställena genom att avsöka skärmen med denna, men det visade sig ta för lång tid (se kap 6.3.1).

5 Användarönskemål

Tidigt under arbetets gång började jag söka efter testpersoner i målgruppen. Det visade sig att det var svårt att få tag i. På vissa håll stötte man bara på sekretessbestämmelser som hindrade kontakt med personer ur målgruppen. Det visade sig också att det var svårt att hitta personer ur målgruppen som kunde tillräckligt om datorer för att komma med bra synpunkter.

Jag var tvungen att börja utveckla programmet utan att ha fått tag i några personer i målgruppen som kunde komma med önskemål. Men dels kunde jag rådfråga sekundär-användare, det vill säga personer som arbetar med personer i målgruppen. Dels ingår jag själv i målgruppen.

5.1 Mina önskemål

Förutom vad jag tagit upp som övergripande mål och ursprungliga idéer, kom jag under arbetets gång fram till flera egna önskemål. Det var viktigt för mig att det skulle vara möjligt att utlösa alla mushändelser med diskreta tangenttryckningar så som: klick, dubbelklick, börja hålla ned musknapp, släpp musknapp, byt aktuell musknapp. En annan sak som var mycket viktig var att det enkelt skulle gå att byta och förändra arbetssätt efter egna önskemål. För ritprogram ville jag få någon metod som simulerade musrörelser så bra som möjligt, dvs använde relativa förflyttningar av muspekaren i önskad riktning.

5.2 Sekundäranvändarönskemål

Vid samtal med sekundäranvändare, till exempel på Certec, kom vissa krav på användargränssnittet fram. En nybörjare måste kunna börja använda programmets enklaste funktioner utan att sätta sig in i alla detaljer. Kryssningsalternativ i menyer skall helst vara kopplade till både text och bild, eftersom olika personer föredrar olika saker för att hitta en kryssruta. Viktiga grafiska objekt måste synas tydligt för att dra uppmärksamheten till sig.

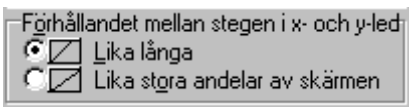


Bild 5.1 Exempel på hur text och bild kompletterar varandra.

6 Programutveckling

6.1 Vidareutveckling av idéer

6.1.1 Enhetliga klicksätt

För att programmet skall vara lätthanterligt för användarna är det bra om så mycket som möjligt är enhetligt för alla metoderna. En sådan sak som gick att genomföra var enhetliga klicksätt. Med klicksätt menar jag allt som går att göra med en mus utöver själva förflyttningen, se tabell 6.1.

Mushändelse	Klick	Dubbelklick	Håll ned/ släpp musknapp	Byt aktiv musknapp
Tangent	5	,	0	/

Tabell 6.1

6.1.2 Varianter på stegvis förflyttning

Den ursprungliga idén med stegvis förflyttning kom, som nämnts tidigare, från Harvard Graphics. Här var det enda sättet att förändra steglängden att dubblera eller halvera den. Detta gjorde att steglängden bara kunde anta värden som var potenser av två. Detta är långt ifrån alltid lämpligt.

Därför bör man dels kunna ställa in steglängden i en meny och dels kunna förändra den genom en tryckning på plus eller minus, för ökning respektive minskning. I menyn skall man kunna välja hur stor förändringsfaktorn *f* skall vara, det vill säga om man önskar dubblering/ halvering eller tredubbling/ få en tredjedel etc. Detta på grund av att förändringsfaktorn fyra är den teoretiskt optimala (se appendix A), medan förändringsfaktor två är mer lätthanterlig om man vill slippa byta riktning, och därmed tangent, för ofta. Det tar ofta betydligt längre tid att trycka på två olika tangenter jämfört med att trycka på samma tangent två gånger. Flera byten av riktning under en musrörelse hämmar dessutom upplevelsen av simulering av kontinuerlig musrörelse.

Om man önskar att snabbt få steglängden till en potens av förändringsfaktorn, så skall man kunna trycka * och önskad exponent.

Denna ursprungliga form av stegvis förflyttning blir effektiv när det är verkligt nödvändigt att simulera musrörelser, till exempel i ritprogram. Även i dataspel med rutor såsom MS röj, Schack, Dam och Othello blir detta effektivt, då man kan ställa in steglängden till att motsvara precis en ruta.

Då det ofta är enklast för användaren att göra de långa hoppen i början av en förflyttning och sedan göra finare och finare justeringar ju närmare målet man kommer, kan man tänka sig att man önskar en minskning av steglängden för varje förflyttning och att steglängden går tillbaka till en lång steglängd (t. ex. en tredjedel av skärmbredden) när man utnyttjar något av klicksätten. Om man dessutom låter musmarkören hoppa till mitten av skärmen efter vart klick, har förändringsfaktor tre och låter stegen i x- och y-led motsvara lika stora andelar av skärmen, istället för att vara lika långa, har vi faktiskt fått rutnätsmodellen.

Bild 6.1 För att ge användaren valfrihet att välja vilka av ovan nämnda alternativ han vill ha, är det lämpligt att man kan kryssa i dessa, var för sig, i en meny. Men samtidigt vill man kanske lätt, genom att ändra inställning på en plats, kunna växla mellan rutnätsmodellen och ”vanlig” stegvis förflyttning. Därför bör dessa alternativ vara åtkomliga att välja direkt.

Det går ej att komma till varje plats på skärmen om man har en automatisk minskning av steglängden med en faktor som är större än tre. Rutnätsmodellen är då överlägset den mest effektiva formen av stegvis förflyttning för att spara tangenttryckningar, i de fall de intressanta ställena inte ligger i ett regelbundet rutnät (se appendix A).

6.1.3 Numrering av intressanta ställen

För att de intressanta ställena skulle bli numrerade var min ursprungliga tanke att man skulle trycka på en funktionstangent för att visa numren på skärmen. Men för att spara tangenttryckningar, så är det mest effektiva att låta denna tangenttryckning också avgöra vad musen skall göra vid det intressanta stället. Alltså låter man här tangenterna [5], [,] och [0] (se tabell 6.1), numrera de intressanta ställena. Dock sker ej själva mushändelsen förrän man slagit in numret på det ställe man avser.

Att mushändelsen skall inträffa direkt då numret är slaget förutsätter att programmet känner till när detta inträffar. Då ett av grundkraven på metoderna var att de inte fick vara tidskritiska, kan man ej utnyttja något tidsavstånd efter senaste tangenttryckningen. Detta innebär att det inte går att ha två nummer på intressanta ställen där den ena sifferströngen är ett prefix av den andra, såsom ”1” är ett prefix av ”10”. Det enklaste sättet att undvika detta är att låta alla nummer ha lika många siffror.

För att åter igen minimera antalet tangenttryckningar, är det bäst att avgöra hur många siffror numren skall innehålla från gång till gång. Är antalet intressanta ställen mindre än elva så numrerar man: 0, 1, 2, 3,… . Vanligast är att antalet ligger i intervallet 11-100 och då numrerar man: 00, 01, 02, 03,…, 09, 10, 11,… . Om antalet överstiger 100, blir det nödvändigt att numrera: 000, 001, 002, …, 099, 100, 101,… .

6.1.4 Hopp mellan intressanta ställen/objekt

I avsnitt 3.2.4 presenteras denna idé som om det alltid var självklart vad som menas med ”närmaste objekt i angiven riktning”. Av bild 6.2 framgår att det krävs en mer preciserad definition av detta begrepp.

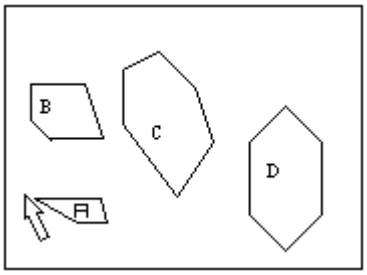


Bild 6.2 Frågan är vilket av objekten A, B, C och D som är det närmaste i höger riktning från pilen, dvs muspekaren.

Den definition som var den mest användbara är också den enklaste. Man ser bara vilket nytt objekt som stöts på först om muspekaren (en pixel bred) vandrar i angiven riktning. I bild 6.2 blir det objekt C. Någon kan invända att det verkar hopplöst att träffa objekt om man bara söker på ett område som är en pixel brett. Detta hade mycket riktigt varit fallet om man sökte efter objekt som själva hade en utsträckning på en eller ett par pixlar. Emedan de flesta intressanta objekt har en mycket större utsträckning än så, finns inga större nackdelar med denna definition.

Entydigheten kan det knappast råda något tvivel om. Om det inte kan anses helt enkelt för användaren att avgöra vilket objekt som kommer att avses, så är det åtminstone enklare än de andra alternativen. Det blev alltså denna definition som användes i programmet.

Man kunde tänka sig att man valde att endast se till vad som ligger närmast med avseende på koordinataxeln som motsvarar angiven riktning. I bilden blir det objekt B. Ett annat tänkbart alternativ är att man tar det objekt på önskat halvplan (i exemplet högra halvplanet till en lodrät linje genom muspekaren) som ligger närmast muspekaren. Detta ger objekt A i bilden. (A, B, C och D ligger alla i högra halvplanet, men A är närmast.)

Varianter på dessa sätt kunde vara att man begränsade området man undersökte till en remsa med en viss bredd eller en sektor med en viss tillåten vinkelavvikelse från önskad riktning. Problemet med dessa varianter är hur användaren skall kunna veta var dessa områden har sina gränser.

En av de främsta fördelarna med hopp mellan intressanta ställen skulle ju vara att man enkelt skulle kunna ta sig mellan intressanta ställen utan att skärmbilden påverkades. Detta får till följd att det är nödvändigt att använda någon entydig definition av ”närmaste objekt i angiven riktning” som gör att användaren, utan hjälp av hjälplinjer eller dylikt, kan avgöra vart musmarkören kommer att hoppa vid en viss knapptryckning.

Dessa två alternativa förslag till definition är inte lämpliga. De är nämligen inte alltid entydiga, då två olika objekt kan ha samma avstånd, både euklidiskt och i angiven riktning. Det är klart att de med tilläggsspecifikationer hade gått att få entydiga. Emellertid är det även krävande för användarens öga att avgöra vilket objekt som är närmast enligt dessa definitioner.

Oavsett vilken definition av ”närmaste objekt i angiven riktning” man använder sig av, så blir det ibland rätt klurigt, för att inte säga helt omöjligt, att komma till ett specifikt objekt. Därför måste denna metod växlas med någon av de andra metoderna när den används.

6.2 Algoritmidéer för numrering av intressanta ställen

När det gäller numrering av intressanta ställen är det nödvändigt att effektivt ta reda på vilka platser på skärmen som skall numreras. Det mest önskvärda hade varit att så gott som automatiskt få en lista över alla de synliga objekten på skärmen och var de var synliga.

Vid första anblicken av funktionerna i MSAA, såg detta fullt möjligt ut. Dock visade sig detta vara omöjligt i praktiken (se 6.3.1). Följande algoritm trodde jag skulle kunna användas:

6.2.1 Lägga alla objekt i en dubbelriktad kantlista

Först skapar man en lista L med alla objekt, sorterade efter djup, så att de bakersta ligger först i L och det främsta sist. Detta gör man genom att anropa Getlist(skrivbordet).

L=Getlist(myobj) {

L=[myobj]

För varje barn b (i djuphetsordning) till myobj så:

L=[L|Getlist(b)] }

Sedan hade man kunnat gå igenom listan L från början till slut och lägga in objekten i en dubbelriktad kantlista[15], med endast vertikala och horisontella kanter. Då man lade in ett nytt objekt skulle man använda följande algoritm:

InputObject(K=gamla kantlistan, N=nya objektets kantlista, U=nya kantlistan) {

1. U:= unionen av K och N.
2. Beräkna alla skärningspunkter mellan de två lodräta sidorna i N och alla vågräta element i K och mellan de två vågräta sidorna i N och alla lodräta element i K.
3. Uppdatera U kring dessa punkter.
4. Gå igenom alla kanter i U och avlägsna de som ligger innanför N.

}

Resultatet av detta hade blivit en dubbelriktad kantlista som motsvarade aktuell skärmbild. Det hade då ej varit någon konst att numrera cyklerna och därmed objekten på skärmen.

Denna kantlista kunde också ha används för hopp mellan intressanta ställen. Man hade bara behövt ta reda på korsningarna mellan önskad stråle och kanterna i listan. Dock visade Getlist sig omöjlig att skapa i praktiken (se 6.3.1).

6.2.2 Hitta övre vänsterhörn genom avsökning av skärmen

Jag fick då återvända till min ursprungliga idé med att avsöka skärmen med hjälp av funktionen *AccessibleObjectFromPoint(p,&pAcc, &varChild).* Då gällde det att hitta ett effektivt sätt att göra detta på.

Det faktum som jag nu noterade var att alla intressanta ställen har minst ett, men inte särskilt många övre vänsterhörn och att dessa platser kunde hittas relativt lätt med följande algoritm:

[15] se M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry - Algorithms and Applications - Second Edition, Springer, 1999, s. 29.

Låt y-koordinaten vandra uppifrån och ned:

OldObj:=nil

Låt x-koordinaten vandra från vänster till höger:

NewObj:=objektet som visas på de aktuella koordinaterna.

Om ej NewObj=OldObj så:

OldObj:=objektet som visas på pixeln över den aktuella.

Om ej NewObj=OldObj så:

Lagra de aktuella koordinaterna som ett övre vänsterhörn.

OldObj:=NewObj

När man har fått tag i alla övre vänsterhörn med ovanstående algoritm, får man gå igenom dessa och hoppa över dubbletter, det vill säga se till att bara numrera ett hörn för varje objekt.

6.3 Problem och begränsningar vid implementation

6.3.1 Microsoft Active Accessibility:s möjligheter och begränsningar

Som tidigare nämnts är Microsoft Active Accessibility (MSAA) en verktygslåda för att göra windowsprogram mer lättillgängliga för personer med funktionshinder. Vid första anblicken såg det ut som att här fanns alla de verktyg och funktioner som kunde tänkas vara användbara för att hitta intressanta ställen på skärmen. Här fanns metoder för att komma åt ett objekts skärmkoordinater, dess barn (till exempel ett fönsters knappar) med mera. Tyvärr visade det sig emellertid att dessa inte var användbara, i praktiken på dagens persondatorer, då det gällde numrering av intressanta ställen. I detta läge var jag tvungen att lägga ner planerna på en fungerade implementation av numrering av intressanta ställen.

Om alla MSAA:s funktioner hade fungerat fullt ut i kombination med alla program i Windows, så hade man lätt kunnat få fram alla objekts koordinater (se 6.2.1). Emellertid är MSAA beroende av att programmen som det skall interagera med är skrivna på ett bestämt standardiserat sätt. Då långt ifrån alla windowsprogram följer denna standard så blev algoritmen i 6.2.1 omöjlig att använda.

Vad som däremot fungerade bättre var anropet: *AccessibleObjectFromPoint(p,&pAcc, &varChild)*. Detta anrop ger för det mesta något relevant svar på vilket objekt som visas på punkten p på skärmen. Då det gäller hopp mellan intressanta ställen så gick denna metod bra att använda för att avsöka utefter önskad stråle.

Min grusade förhoppning var att det även skulle gå att utnyttja denna funktion för numrering av intressanta ställen enligt avsnitt 6.2.2. Då det inte är rimligt att användaren skall vänta längre än två sekunder på att numreringen skall utföras och man bör kontrollera minst 64 x 48= 3072 ställen på skärmen för att ej missa något intressant ställe, så får varje kontroll med denna funktion i genomsnitt ta 0,00065 sekunder.

Jag mätte då funktionstiden för *AccessibleObjectFromPoint* på två olika datorer. På en processor på 450 MHz mätte jag upp genomsnittstider på mellan 0,0026 och 0,026 sekunder. På en 1800 MHz processor erhöll jag resultat på mellan 0,0012 och 0,0054 sekunder. Vid dessa test kunde jag också konstatera att det ej gick lätt att kartlägga vad denna funktionstid beror på. Jag fick variationer på upptill en faktor två på utåt sett identiska test.

Då det visade sig att man på dagens vanliga persondatorer knappast kommer i närheten av den korta funktionstid som hade krävts för en snabb numrering av intressanta ställen, så var denna idé omöjlig att få att fungera i praktiken.

6.3.2 Problem med grafik på skärmen

I alla mina metoder hade det varit önskvärt att kunna rita grafik på skärmen som inte uppfattades som något objekt som innehade fokus. Tyvärr har jag i detta examensarbetet misslyckats med att åstadkomma detta. Framför allt hade det underlättat om det gick att rita ut rutnätet i rutnätsmodellen. I stegvis förflyttning hade man kunnat markera var muspekaren hamnar om den flyttas i respektive riktning, dels med innevarande steglängd och dels med de steglängder som är potenser av förändringsfaktorn och därigenom går lätt att komma åt (se Appendix B). Vid hopp mellan intressanta ställen, hade man kunnat leta upp och markera vilka ställen som musmarkören kommer att hoppa till i respektive riktning.

Skenbart kan detta se enkelt ut att ordna. Via koden:

```
function winDesktoCanv: TCanvas;
var DC: HDc;
begin
   DC := getWindowDC ( GetDeskTopWindow );
   result := TCanvas.Create;
   result.Handle := DC;
end;
```

får man tillgång till skärmens kanvas (rityta). Denna kan man sedan både rita och skriva på. Problemet kommer då man önskar att se till att det man har ritat verkligen försvinner, utan att lämna några spår efter sig.

Min första tanke var att använda invertering av färger. Då man ritade ut något, skulle man göra detta genom att invertera färgen på de pixlar man ville markera. När man skulle ta bort sin grafik, skulle man bara behöva invertera tillbaka färgen. Tanken var att använda en egen inverteringsfunktion som såg till att varje färg hade en unik invers som dessutom bildade en klart synlig kontrast med färgen. Problemet med denna metod, liksom med tanken att spara undan färgen man ritar över, är att andra program kan ändra på grafiken på skärmen utanför programmets kontroll mellan utritande och borttagande av hjälpgrafik.

Vad som hade behövts för att undvika detta är att använda ett transparent fönster. Sådana fönster har en genomskinlig bakgrundsfärg. Det vill säga att bakgrunden släpper igenom bakomvarande objekt. Tekniken hade varit att skapa ett transparent fönster som täcker hela skärmen och alltid ligger främst. På detta fönster kunde programmet ha skapat sin grafik och sedan tagit bort den med hjälp av den genomskinliga bakgrunds-färgen.

På de fönster som man skapar i Delphi 6 finns det en flagga, *TransparentColor*, att sätta till *True*, om man vill att fönstret skall bli transparent. Problemet var att denna egenskap bara fungerar i Windows 2000 och Windows XP. Då de flesta privatanvändare av Windows ej har denna typ av Windows, och jag inte hade omedelbar tillgång till detta under arbetets gång, så tyckte jag inte att detta var någon framkomlig väg.

En annan tänkbar möjlighet att skapa ett transparent fönster på, är att göra det i en dll i ett annat språk och sedan anropa dessa metoder från Delphi. Inom detta arbetes ram fanns tyvärr inte tid att undersöka denna möjlighet. Detta gjorde att jag fick ge upp tanken på implementation av hjälpgrafik.

7 Användartester

Som nämnts redan i kapitel fem, har det varit svårt att få tag i testpersoner ur den egentliga målgruppen. När jag trott att jag äntligen har fått tag i rätt person, så har oftast en sjukdom eller något annat kommit emellan. En person, förutom jag, med rörelsehinder har dock fått testa det färdiga programmet. I övrigt har jag fått hålla mig till sekundäranvändartester (se 5.1), test av en utomstående systemutvecklare och test på mig själv.

7.1 Synpunkter från en webbtekniker med rörelsehinder

Den person med rörelsehinder i händerna som har testat det färdiga programmet var själv webbtekniker på en teknisk högskola. Han uttryckte att det var ett bra program för att styra musmarkören då man har problem med finmotoriken i handen. Det var en mycket bra produkt som nog kommer hjälpa många personer med funktionshinder, tyckte han. Vidare erbjöd han sig att hjälpa till med lanseringen av programmet. Dock var det inget program för honom personligen, då han klarar sig bra med att använda en joysticksmus (se 2.4.1).

7.2 Sekundäranvändares synpunkter

Programmet har även testats av personal på Dahjm som är ett REDAH-center (se 2.2) beläget i Lund och på Furuboda KompetensCenter[16]. De ansåg att HotMouse är ett bra komplement till de tillgängliga produkter som finns på marknaden. Nya finesser som de tre valmöjligheterna till förflyttning som inte finns på marknaden idag uppskattades. Justeringarna som kan göras direkt i Windows-miljö, alternativt i program, utan att behöva går ur programmet ansågs vara en styrka.

Programmet upplevdes flexibelt att ställa in så att det passar olika användare, men det passar bäst till den som har goda eller mycket goda kognitiva och perceptuella förmågor. De många valmöjligheterna i programmet gör att det krävs att användaren kan tänka i steg, både framåt och bakåt.

För den som motoriskt klarar att styra ett numeriskt tangentbord med direktpekning ansågs det vara ett effektivt sätt att styra datorn. Man kunde också tänka sig att det, med ytterligare anpassning i form av mjukvara eller hårdvara, skulle användas av personer som använder en eller två kontakter.

[16] http://www.furuboda.se/resurscenter/resurs.asp

Kortkommando för att växla läge samt att få fram en översikt över de olika tangentbordskommandona efterlystes. Vid stegvis förflyttning vore det värdefullt om användaren kunde välja bort att markören tog det sista steget ut till skärmkanten eftersom det rubbar steglängen och det mönster som markören följer. Detta vore särskilt intressant för den som använder en rutbaserad kommunikationsprogramvara.

7.3 Synpunkter från en utomstående systemutvecklare

Jag har också låtit en utomstående systemutvecklare testa programmet i cirka två timmar. Han jobbar på ett företag som inte har någon anknytning till personer med funktionshinder. Som flera andra flitiga datoranvändare föredrar han att använda tangentbordet framför musen ibland. För sin egen del tyckte han att det verkade användbart vid de tillfällen man vill kunna kontrollera datorn med en hand. Det kan man vilja om man till exempel håller en telefonlur i andra handen.

De olika metoderna och inställningsmöjligheterna upplevdes komplettera varandra på ett bra sätt. Dock borde det vara lättare att växla mellan olika styrsätt. Stegvis förflyttning med faktor två och tre var det styrsätt som var enklast att använda. Faktor tre tycktes ge en lagom förändring av steglängden. Det uppskattades att man snabbt kunde förflytta sig över hela skärmen genom att förändra steglängden med hjälp av plus- och minustangenten.

Anledningen till att han föredrog stegvis förflyttning framför rutnätsmodellen var troligen att han genomförde testet i huvudsak i ritprogrammet Paint och där önskade en relativ förflyttning. Hopp mellan intressanta ställen upplevdes fungera väl inom vissa verktygsfält. Utanför dessa fungerade det mindre bra, då ej alla intressanta ställen upptäcktes.

8 Resultat och slutsatser

8.1 Programmet

Det mest påtagliga resultatet av detta examensarbete är den utvecklade programvaran. Programmet har fått namnet HotMouse och dess funktioner finns i korthet beskrivna i appendix B. Min förhoppning är att, efter eventuellt ytterligare förbättringar, få ut det på marknaden.

8.2 Egna effektivitetsvinster

Jag har genom att skapa detta program fått upp hastigheten i mitt musanvändande mycket. Hur mycket är dock svårt att uppskatta. Detta beroende på att jag parallellt med examensarbetet har gjort betydande framsteg rent motoriskt med hjälp av fysisk träning. Det ger därför ingen rättvisande bild att jämföra mitt datoranvändande före och efter examensarbetet. Vad jag kunde jämföra var min hastighet i dagsläget med mitt program respektive Windows inbyggda musstyrning.

För att testa detta på ett så objektivt sätt som möjligt, skapade jag ett testprogram. Testen gick ut på att klicka tio gånger på en knapp som flyttade sig till en slumpvis vald plats efter vart klick. För att få det så rättvisande som möjligt, använde jag samma

slumpvist utvalda platser vid båda testen. Då det kan vara en liten fördel att ha sett platssekvensen innan började jag med testet av mitt eget program, för att vara säker på att inte gynna detta. Jag fick här med rutnätsmodellen en tid på 71 sekunder, alltså 7.1 sekunder per förflyttning. Vid testet med Windows inbyggda musstyrning uppmättes en tid på 101 sekunder, alltså 10.1 sekunder per förflyttning. Det betyder att mitt program har åstadkommit åtminstone en 30-procentig förbättring för min egen del.

Emellertid var skillnaden större då jag precis hade skapat programmet. En uppenbar vinst jag erövrade direkt när programmet var användbart, var att jag kunde börja spela Dam med okända motspelare i realtid via internet. Tidigare hade motspelarna hunnit tröttna och lämna spelet innan jag hunnit flytta min pjäs. Tyvärr har jag inga uppmätta siffror på hur mycket snabbare mitt musanvändande blev när jag kunde börja använda mitt program. Dock vet jag att skillnaden då var betydligt större än dagens 30 procent. Anledningen till att skillnaden har krympt är att jag, genom min fysiska träning, framför allt har fått bättre reaktionsförmåga (se 2.4.1).

9 Idéer för vidareutveckling

9.1 Förverkliga ouppnådda mål

Som har framgått tidigare i rapporten finns det en rad av idéer som jag ej har lyckats med att skapa en fungerande implementation av. Framförallt gäller det ”numrering av intressanta ställen” och hjälpgrafik.

Då det gäller hjälpgrafik, vet jag att det är möjligt att skapa en fungerande implementation. Detta har till exempel VoiceMouse och Dragon NaturallySpeaking (se 2.4.3) lyckats med. Inom detta examensarbetets ram har jag tyvärr misslyckats med att få detta att fungera. Dock tänker jag försöka att åstadkomma detta efter denna redovisning.

När det gäller numrering av intressanta ställen anser jag mig däremot ha uttömt dagens möjligheter. För att åstadkomma en fungerande implementation av detta, tycks man få vänta på antingen mycket snabbare datorer eller att alla program anpassas till att fungera tillsammans med MSAA.

En tredje sak som fanns med i mina ursprungliga idéer, men som ej har funnits tidsutrymme att vidareutveckla är simulering av mjuka musrörelser med hjälp av beizerkurvor. Jag har medvetet prioriterat bort det, då det hade krävs tillgång till hjälpgrafik för att man som användare skulle ha någon praktisk användning av detta. Möjligen kan jag ge mig på denna uppgift, om jag framöver får hjälpgrafiken att fungera.

9.2 Praktiska förbättringar

Detta examensarbete har i huvudsak inriktat sig på att ta fram, implementera och analysera olika metoder för att styra musmarkören via tangentbordet. Det innebär att det inte har funnits mycket utrymme för att utveckla kontaktytan för att ändra inställningar eller byta mellan olika styrsätt.

En av allvarligaste bristerna på detta område är att inställningsfönstret inte går att minimera via den vanliga minimeringsknappen uppe till höger på windowsfönstret. Klickar man på denna så händer ingenting. Jag vet att det har med användningen av

TrayIcon (se 4.3) att göra, men inte hur det kan ordnas till. I nuläget får man istället välja ”Fönster” och sedan ”Minimera” i fönstermenyn. Det finns också en risk att en användare som vill få bort inställningsfönstret helt enkelt stänger det. Det är bara det att då slutar hela programmet att fungera. Detta borde naturligtvis också åtgärdas.

Ytterligare en tänkbar förbättring är att programmet skulle kunna spara inställningar. Det hade varit en fördel om de inställningar som gäller när programmet eller datorn stängs av, också gäller nästa gång programmet startas. Användare skulle också kunna spara egna favoritinställningar i filer för att lättare kunna växla mellan dessa.

Sist men inte minst bör det finnas en utförlig manual som beskriver programmets funktioner. I denna beskrivning skall bland annat finnas en bild på det numeriska tangentbordet, där knapparnas funktioner förklaras. Detta för att annan hänvisning till det numeriska tangentbordet kan vara missvisande då symbolerna på dessa tangenter kan variera mellan olika tangentbord. Denna manual bör finnas som hjälpfil i programmet.

Appendix A – Värstafalls-analyser av styrsätt

När det gäller förflyttning av muspekaren till valfri pixel på skärmen, har jag utvecklat ett par olika metoder. Här vill jag reda ut vilken/ vilka varianter av metoderna som kräver minst antal tangenttryckningar. Jag har valt att analysera värstafallen, då detta ger minst omfattande uträkningar och ändå ger relevanta resultat.

För att inte få alldeles för komplicerade uträkningar så granskar vi styrsätten för förflyttning i endast en dimension. Detta bör ej inverka på resultaten, då man kan sköta förflyttningarna i x- och y-led parallellt med hjälp av de diagonala riktningarna.

a= avstånd

d= tillåten felmarginal

s= skärmbredd

f= förändringsfaktor i stegvis förflyttning

Rutnätsmodellen

Här minskas rutbredden till en tredjedel för varje tangenttryckning vilket ger $\lceil \log_3(s/d) \rceil = \lceil 0.91\ln(s/d) \rceil$ tangenttryckningar i värsta fall.

Stegvis förflyttning

Vi antar först att man utgår ifrån en steglängd som ej skiljer sig från a med mer än en faktor *f*. För att komma ned till önskad noggrannhet måste vi passera $\lceil \log_f(a/d) \rceil$ nivåer. För varje nivå, kan man i värsta fall behöva flytta markören $\lfloor f/2 \rfloor$ steg åt ena eller andra hållet. Sedan måste man trycka på en tangent för att komma till nästa nivå.

Om *f* är udda så täcker man in varje plats på nivån genom att ta $0 \leq n \leq (f-1)/2$ steg till höger eller vänster från en bestämd plats på ovanliggande nivå. Detta på grund av att varje heltal går att skriva som $a*f \pm n$ där *n* är ett naturligt tal $\leq (f-1)/2$ och *f* är ett givet udda tal och *a* är ett unikt heltal, ty $a*f+(f-1)/2+1=(a+1)*f-(f-1)/2$

Om *f* är jämt så kan *n* högst behöva vara *f*/2 detta medför dock en valfrihet i val av *a*, då $a*f+f/2=(a+1)*f-f/2$. Detta medför att om man är tvungen att ta *f*/2 steg på en nivå så har man en valfrihet i ovanliggande nivå och kan därför bara behöva flytta högst *f*/2-1 steg i denna. Man behöver alltså göra högst $f/2+f/2-1=f-1$ förflyttningar under två nivåer.

Oavsett om *f* är udda eller jämt så krävs det alltså igenomsnitt högst (*f*-1)/2 förflyttningar per nivå.

Alltså kan det krävas $((f-1)/2+1)=(f+1)/2$ tangenttryckningar per nivå.

Sammanlagt antal tangenttryckningar kan uppskattas till:

$\log_f(a/d)*(f+1)/2=\ln(a/d)*((f+1)/2))/\ln f=\ln(a/d)*g(f)$

$g(f)=(f+1)/(2\ln f)$

$g(f \to 1_+) \to +\infty$

$g(2)=2.16$

$g(3)=1.82$

$g(4)=1.80$

$g(5)=1.86$

$g'(f)=0.5(1/\ln f+(f+1)*(-1/(\ln f)^2)/f)=0.5(f\ln f-f-1)/(f(\ln f)^2)$

då $f>5$ så $g'(f)>0.5(1.6f-f-1)/(f(\ln f)^2)=(0.3f-0.5)/(f(\ln f)^2)>(0.2f+0.5-0.5)/(f(\ln f)^2)>0$

ger att g är växande då $f>5$ => g(4) är det minsta värde g antar för något heltal *f*. Alltså är faktor 4 den teoretisk optimala under givna förutsättningar. Då kan man uppskatta antalet tangenttryckningar med $1.80\ln(a/d)$. Dock ser vi också att den teoretiska skillnaden på användning av förändringsfaktorerna tre och fyra bara cirka en procent, vilket gör att de i praktiken är lika effektiva.

Rutnät vs stegvis förflyttning

$0.91\ln(s/d)>1.80\ln(a/d) \Leftrightarrow$

$(s/d)^{0.91}>(a/d)^{1.80} \Leftrightarrow$

$s/d>(a/d)^{1.98}$

vilket säger att om förflyttningsavstånden inte har något direkt samband med varandra så är rutnätsmodellen att föredra, då de flesta förflyttningar är längre än roten ur skärmbredden, med felmarginalen som enhetsmått.

Appendix B – Sammanställning av HotMouse's funktioner

HotMouse är ett program som är till för att styra musmarkören med hjälp av diskreta tangenttryckningar på det numeriska tangentbordet, då numlock är på.

Det finns i huvudsak tre fungerande metoder i programmet: stegvis förflyttning, rutnät och hopp mellan intressanta ställen. Man byter styrsätt i inställningsfönstret som man får upp genom att trycka **. I detta fönster finns även andra inställningsmöjligheter.

För alla tre styrsätten gäller:

Tangenter	funktion
1-4,6-9	flyttar muspekaren i den riktning som tangenten har i förhållande till 5:an.
5	enkelklick med aktuell musknapp.
,	dubbel----------”----------
0	börja/sluta hålla ned aktuell musknapp.
/	växla aktuell musknapp.
**	tar fram inställningsfönstret.

För hopp mellan intressanta ställen går musmarkören i angiven riktning tills den stöter på något nytt objekt på skärmen. Den känner dessvärre ej av alla intressanta ställen.

För de två andra styrsätten går musmarkören en bestämd steglängd i önskad riktning. Denna steglängd ökas respektive minskas med en förändringsfaktor vid tryck på + respektive -. Denna förändringsfaktor går att ställa in i inställningsfönstret. För att rutnätsättet skall fungera bör den var tre.

Om man vill sätta steglängden direkt till en potens av förändringsfaktorn så kan man trycka * följt av önskad exponent.

Skillnaden mellan stegvis förflyttning med förändringsfaktor tre och rutnät är att rutnätsmodellen minskar steglängden automatiskt efter varje hopp och återvänder till skärmens centrum, samtidigt som den sätter steglängden till en tredjedel av skärm-bredden efter musklick. Dessutom är här stegen inte lika långa i x- och y-led utan lika stora andelar av skärmen. Alla dessa alternativ går också att ställa in var för sig i inställningsfönstret.

Rutnät heter rutnät för att i praktiken så kan man se det som att programmet delar in skärmen i tre gånger tre rutor som man väljer bland genom att antingen trycka minus för mittenrutan eller siffran i önskad riktning. Sedan delas önskad ruta in på samma sätt. Tyvärr har jag inte kommit på något bra sätt att visa rutnätet på skärmen.

Systemkrav: Windows 98/ Me