UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matija Lovrić

VPELJAVA GEST Z MIŠKO IN NADGRADNJA FUNKCIONALNOSTI KLASIČNEGA UPORABNIŠKEGA VMESNIKA

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Franc Solina

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00113/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATIJA LOVRIĆ**

Naslov: **VPELJAVA GEST Z MIŠKO IN NADGRADNJA FUNKCIONALNOSTI KLASIČNEGA UPORABNIŠKEGA VMESNIKA**

INTRODUCING MOUSE GESTURES AND FUNCTIONALITY UPGRADES INTO CLASSIC USER INTERFACE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi so predstavljene razširitve klasičnih grafičnih uporabniških vmesnikov v smislu gest z miško, ki nadomeščajo njihove pomanjkljivosti. Opisana je optimalna logika konfiguriranja razširitev, ki pripelje do maksimalne intuitivnosti, poenostavitve in pospešitve dela z računalnikom ter uporabniku omogoči maksimalno produktivnost in prihranek časa. Poleg gest z miško kot razširitve je predstavljen tudi skriptni jezik, s katerim obogatimo in povečamo spekter funkcionalnosti, ki ga vmesnik ponuja uporabniku.

Mentor:

prof. dr. Franc Solina

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Matija Lovrić,

z vpisno številko 63020332,

sem avtor diplomskega dela z naslovom:

VPELJAVA GEST Z MIŠKO IN NADGRADNJA FUNKCIONALNOSTI KLASIČNEGA UPORABNIŠKEGA VMESNIKA

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom (naziv, ime in priimek)

  prof. dr. Franc Solina

  in somentorstvom (naziv, ime in priimek)

  as. dr. Luka Šajn

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

Zahvala

Moja največja zahvala gre mojim staršem, ki so mi brezpogojno omogočili ta študij in mi stali ob strani vse do konca. Hvala oči, da si me naučil vstati po padcu. Hvala mama, da si mi pomagala in me naučila vztrajnosti.

Hvala tudi tebi moja babica, da si vedno verjela vame in na nekakšen čuden način jasno videla moj cilj tudi v tistih trenutkih, ko sem sam zašel v temo. Žal mi je, da nisi dočakala tega dneva, pa vendar vem, da si v svetlobi tudi zdaj in si zelo ponosna name.

Najlepša hvala mentorju prof. dr. Francu Solini in somentorju dr. Luki Šajnu za pomoč in podporo.

Velika hvala mojemu drugu in sošolcu Luki Mrovljetu za skrbno in nesebično pomoč.

Kazalo

POVZETEK **0**

1 UVOD **3**

2 KRATEK PREGLED ZGODOVINE **6**

2.1 PRAZGODOVINA 6

2.2 ZAČETEK GUV-A IN "MATI VSEH PREDSTAVITEV" 7

2.3 RAZISKOVALNI CENTER PALO ALTO 10

2.4 SMALLTALK 13

2.5 APPLE 15

2.6 ŠIRJENJE GUV REVOLUCIJE 16

2.6.1 X Window System *17*

2.6.2 Microsoft *18*

3 ZAKAJ GESTE Z MIŠKO? **20**

4 METACITY **24**

4.1 PREDNOSTI UPRAVLJALNIKA OKEN METACITY 24

4.2 SLEDENJE KONCEPTU AKORDIČNE TIPKOVNICE 26

4.3 NAMESTITEV BLIŽNJIC 27

5 GESTE V GNU/LINUXU **32**

5.1 EASYSTROKE 32

5.2 ZGLED PROGRAMSKEGA VMESNIKA 32

5.3 KONFIGURIRANJE GEST 39

5.3.1 Enostavne geste *39*

5.3.2 Napredne geste *46*

6 GESTE V WINDOWSIH **54**

6.1 STROKEIT 54

6.2 ZGLED PROGRAMSKEGA VMESNIKA 55

6.3 NASTAVLJANJE DEJANJ 59

7 AUTOIT 69

7.1 PREMIKANJE OKNA 70

7.2 KRČENJE IN VEČANJE OKNA 72

7.3 ROČNO PREMIKANJE IN SPREMINJANJE VELIKOSTI OKNA 75

7.4 POMIKANJE VSEBINE 79

7.4.1 Samodejno pomikanje 80

7.4.2 Ročno pomikanje 83

7.4.3 Navijajoče pomikanje 83

7.4.4 Več funkcionalnosti znotraj ene skripte 85

8 SKLEP 89

9 VIRI 91

Seznam uporabljenih kratic

GUV – grafični uporabniški vmesnik

WIMP (ang. "window, icon, menu, pointing device") – oblika načrta uporabniškega vmesnika, zasnovana na interakciji uporabnika s pomočjo oken, ikon, menijev in kazalnih naprav

HCI (ang. Human-computer interaction) – preučevanje, planiranje in načrtovanje interakcije med človekom in računalnikom

RISC (ang. Reduced instruction set computing) – vrsta strategije pri načrtovanju centralnih procesnih enot

KISS (ang. "Keep it simple, Stupid!") – načelo načrtovanja, ki predpisuje enostavnost kot ključni cilj in se izogiba nepotrebni zapletenosti

COM (ang. Component Object Model) – Microsoftov standard binarnega vmesnika za sestavne dele programske opreme

DLL (ang. Dynamic link library) – Microsoftova izvedba koncepta deljene knjižnice

LIFO (ang. Last In, First Out) – okrajšava za "zadnji noter, prvi ven", ki opisuje odnos med vrstnim redom vstavljanja in vrstnim redom obdelave elementov neke podatkovne strukture

POVZETEK

Namen diplomske naloge je analizirati značilnosti in trenutne pomanjkljivosti obstoječih popularnih grafičnih uporabniških vmesnikov v najpogostejših operacijskih sistemih osebnih računalnikov v sferi človeško-računalniške interakcije (HCI) in predložiti alternativno smer v njihovem razvoju, ki jo omogočajo geste z miško.

Vsebina je razdeljena na dva dela. Prvi razpravlja o možnih upravljalniških razširitvah znotraj operacijskega sistema GNU/Linux, drugi pa znotraj Windowsa. V GNU/Linuxu so predstavljene možnosti upravljalnika oken Metacity in potem povezane z gestami, ki jih omogoča program Easystroke. Za geste v okolju Windows je poskrbel program StrokeIt. Njegove funkcionalnosti in splošno dostopne možnosti celotnega grafičnega uporabniškega vmesnika operacijskega sistema so na koncu razširjene in obogatene s pomočjo skript, narejenih v programskem jeziku AutoIt.

Ključne besede:

vmesnik, geste, miška, upravljanje, nadzor, intuitiven, razširitev, nadgradnja, akordična tipkovnica, kognitivna obremenitev, Easystroke, StrokeIt, AutoIt

Summary

In addition to analyze the characteristics and present deficiencies in the Human-Computer interaction (HCI) area of the existing and popular graphical user interfaces of the most common operating systems in personal computers, the goal of this thesis is to propose an alternative path in their evolution, made feasible by using mouse gestures.

The contents consist of two parts. First one disserts possible control upgrades within the GNU/Linux operating system and the second one explores that possibility in Windows. The GNU/Linux part presents hidden features of the Metacity window manager first, then binds them with gestures provided by the Easystroke program. The task of providing mouse gestures inside Windows environment is given to the StrokeIt program. In the end, its functionalities as well as the set of commonly available features of the operating system's whole graphical user interface are expanded and enriched with the help of scripts in the AutoIt computer language.

Keywords:

interface, gestures, mouse, control, intuitive, upgrade, chording keyboard, cognitive load, Easystroke, StrokeIt, AutoIt

1 UVOD

"Mnogo ljudi meni, da se je grafični uporabniški vmesnik (v nadaljevanju GUV) skozi čas le malo spreminjal z manjšimi razlikami v osnovni koncepciji namizja, okna, miške, ikon in kazalčka od originala na Lisi 1983. To je najbrž posledica navad uporabnikov, ki so se privadili na stil grafičnega vmesnika Lise/Macintosha in drugih projektov, ki so temu konceptu sledili in izdelali GUV po istem kopitu. A sodeč po skrajno dolgem razvoju GUV-a, ki sega od njegovih skromnih začetkov do danes, in ob dejstvu, da se je količina prodanih PC-jev samo v 90. letih eksponentno večala, je verjetno pravilneje gledati na GUV kot na počasno evolucijo proti idealnemu vmesniku. Čeprav se je poskušalo na radikalne načine spremeniti pristop pri komuniciranju z GUV-om (na primer Sunov Looking Glass prikaz in Microsoftov raziskovalni projekt 3D uporabniškega vmesnika), se zdi, da je do vsesplošne razširjenosti teh sprememb še daleč.

Medtem ko čakamo na prihodnjo različico Windowsa in Mac OS X-a, lahko spoznamo, da čeprav je še veliko prvotne funkcionalnosti GUV-a ostalo nespremenjene, je potencial za dodajanje novih značilnosti in načinov interakcije neskončen." [1]

Začetki GUV-a so povezani z Douglasom Englebartom in letom 1962. Njegove ideje in vizije računalnika, ki ni nadomestek človeka, temveč razširitev človeškega uma, so v tistem času prispevale k dih jemajočim in nekoliko begajočim inovacijam v tehnologiji in načinu dojemanja računalnika nasploh. Englebart je s svojimi rešitvami utiral pot novi vrsti računalnika: digitalnemu osebnemu računalniku. Njegovo delo so nadaljevali in izpopolnili Ivan Sutherland, Alan Kay, Jef Raskin in mnogi drugi prodorni umi, večinoma zbrani v razvojnih ekipah, kot je tista v Xeroxu in kasneje v Applu. Pomembno je vedeti, da GUV, kakršnega imamo danes na osebnih računalnikih, ni proizvod samo enega človeka oziroma samo ene ekipe. Neustrezno bi bilo reči, da je Xeroxov raziskovalni projekt PARC izumil GUV oziroma da gre popolna zasluga za današnji GUV Applu, saj je GUV nastajal počasi in postopno, s prispevki mnogih ljudi skozi njegovo razvojno zgodovino od leta 1962 pa vse do danes. Ob tem se lahko porodi vprašanje, zakaj se v vseh teh 49 letih še vedno nismo odmaknili od WIMP ("window, icon, menu, pointing device") modela upravljanja GUV-a. Ali je res, da privajenost uporabnika na vnaprej določen tip interakcije z GUV-om, ki temelji na vizualnih gradnikih, zavira ali celo onemogoča nadaljnji razvoj vmesnika k novim, bolj intuitivnim ravnem, ki so bolj prilagojene naravnemu človeškemu modelu interakcije z okolico? To je vprašanje, s katerim se bom ukvarjal v tem delu. Skušal bom

najti odgovor nanj in na koncu predstaviti upravljavsko razširitev Windowsovega vmesnika, ki presega dosedanje ustaljene okvire brez uvajanja novih pomanjkljivosti ter v celoti izpolnjuje naslednje pogoje:

1. visoka dostopnost,
2. nizka cena,
3. intuitivnost,
4. enostavnost uporabe,
5. hitrejše in enostavnejše delo na računalniku.

Z **visoko dostopnostjo** mislimo na možnost, da si običajen uporabnik brez težav priskrbi razširitev na javno dostopnem mestu – medmrežju. Programska oprema, ki je visoko dostopna, se lahko prenese z medmrežja brez predhodne avtorizacije ali izpolnjevanja posebnih zahtev.

Nizka cena zahteva minimalno porabo denarja, ki ga mora imeti uporabnik, da si priskrbi, vgradi in začne uspešno uporabljati razširitev.

Intuitivnost se nanaša na samo upravljavsko rešitev, ki nadomesti ali izpopolni dosedanje oblike upravljanja na takšen način, da zmanjšuje kognitivno obremenjenost uporabnika pri delu. Intuitivnost je treba razlikovati od seznanjenosti.

Enostavnost uporabe izhaja iz dobrega načrta in dobre izvedbe rešitve. Končnim uporabnikom se pri tem ni treba posebej seznanjati z vmesnikom, kar prispeva k občutnemu prihranku časa za privajanje na nov sistem upravljanja. Ta zahteva implicira tudi najmanjše možno število korakov za izvajanje posameznih (kompleksnejših) tipov operacij.

Hitrejše in enostavnejše delo na računalniku je poglavitni cilj, h kateremu težimo. To je pozitivni preskok od predhodnega osnovnega vmesnika operacijskega sistema. Nov vmesnik, ki nastane z vgraditvijo razširitve, mora biti enostaven za delo in omogočiti uporabniku časovni prihranek pri izvajanju standardnih operacij v GUV-u, kot so premiki, skrčitve, spremembe velikosti in zapiranja okna, pa tudi nekih kompleksnejših operacij, kot so kopiranje datotek,

upravljanje funkcij za aplikacije itd. Pri takšni povečani hitrosti upravljanja s sistemom ostane uporabnik skoncentriran na glavne naloge pri delu z računalnikom, hitreje pride do rezultata in ne izgublja časa za kompleksno sintakso upravljalnega sistema, kar je še posebej opazno pri uporabniku začetniku.

Treba je razmisliti o obstoječih novih radikalnih preobratih v razvoju tehnologije HCI (vizualno razpoznavanje kretenj telesa, na dotik občutljivi zasloni idr.), njihovi uporabnosti in vplivu na daljni razvoj. Treba je tudi predstaviti do zdaj zanemarjene prednosti vmesnika z gestami, konkretno na primeru gest z miško. Za ta namen bosta razložena procesa konfiguriranja in uporabe programske opreme za geste z miško v operacijskem sistemu GNU/Linux in Windows XP. V zvezi s tem bodo predstavljeni tudi primeri širitve razpoložljivega funkcionalnega spektra Windows GUV-a s pomočjo skriptnega jezika AutoIt.

2 KRATEK PREGLED ZGODOVINE

Da bi razumeli koncept GUV-a, njegove idejne začetke in smer razvoja, ki je privedla do svojevrstne stagnacije danes, je treba pogledati razvojni tok od samih začetkov.

2.1 Prazgodovina

Kot mnoge druge inovacije in izumi so tudi preboji v računalništvu začeli svojo pot z idejami še iz časov, ko je bila tehnologija za njihovo realizacijo v povojih ali pa sploh še ni obstajala. To je trend, v katerem radikalne (in pogosto od javnosti zasmehovane in nerazumljene) ideje nastajajo v neomejeni domišljiji tehnoloških jasnovidcev, ki si zamislijo znanstvenofantastične stroje prihodnosti, ki so z zamislijo stopili na dolgo pot k njihovi daljni realizaciji. Ne rečemo zaman, da je vse mogoče.

GUV tako nastaja kot (v tistem času veliko stvari) plod domišljije peščice ljudi, med katerimi je tudi Vannevar Bush. Ta inženir elektrotehnike z inštituta MIT je znan ne le kot glavni organizator projekta Manhattan, temveč tudi kot avtor ideje o stroju, imenovanem "Memex", o katerem se je pisalo v zgodnjih 30. letih 20. stoletja. Memex je bil sestavljen iz dveh miz z dvema grafičnima zaslonoma z upravljanjem na dotik, s tipkovnico in priključenim skenerjem. Vloga stroja je bila, da omogoči ljudem pristop do vsega znanja človeštva s pomočjo povezav, ki so bile zelo podobne današnjim hiperpovezavam z interneta. Ker digitalni računalnik tedaj še ni bil izumljen in posledično ni obstajalo dovolj znanja, ki bi lahko omogočilo realizacijo tako napredne tehnologije, je Busheva ideja padla v začasno pozabo vse do začetkov digitalnega računalnika okoli leta 1937.

"Druga svetovna vojna je prinesla veliko motivacije in sredstev za izdelavo programskih računskih strojev za vse, od izračunavanja tablic za artilerijsko streljanje do razbijanja sovražnikovih tajnih šifer." [1]

Leta 1945 je Bush ponovno spregovoril o svojih starih idejah v članku z naslovom "Kot bi si lahko mislili" (angl. As We May Think). Članek je bil izdan v časopisu Atlantic Monthly, ki je

prišel v roke mlademu Douglasu Englebartu in v njem prebudil navdih, da poskusi izdelati stroj iz Busheve domišljije.

2.2 Začetek GUV-a in "Mati vseh predstavitev"

Douglas C. Englebart je ameriški izumitelj in eden od pionirjev interneta. Po služenju v ameriški mornarici na Filipinih v II. svetovni vojni kot radarski tehnik se je, navdahnjen z Bushevim člankom, vrnil v rodno deželo Oregon v ZDA in tam diplomiral iz elektrotehnike. Redno zaposlitev je dobil v Naci (predhodnik Nase), kjer je delal med letom 1951. Tistega leta je v njegovem življenju prišlo do velike prelomnice, potem ko je doživel osebno razsvetlitev in zaključil, da dobra izobrazba in dober posel ne moreta biti njegov končni cilj. Odločil se je narediti korake, ki bi naredili svet boljše mesto za vse ljudi. Treba je bilo zbrati ljudi, ki so bili sposobni najti učinkovite rešitve za probleme, ki pestijo vse, in povezati človeške ume v močno mislečo silo. Pri tem je kot sredstvo za izvedbo tega cilja sprejel digitalni računalnik. Spomnil se je Bushevega članka iz leta 1946 in začel načrtovati izgradnjo stroja, podobnega Memexu. Izkušnje v mornarici so ga spomnile na radarski zaslon in takoj je spoznal, da bi morale biti informacije prikazane in preverjene na takšnem zaslonu. A šel je še korak dlje in si zamislil delovne postaje s takšnimi zasloni, na katerih bi "intelektualni delavci" zbirali kolektivno znanje, nato pa skakali od podatka do podatka v informacijskem prostoru in izbirali tiste, ki so pri skupnem delu potrebni za reševanju poljubnega problema. Takšna vizualizacija in upravljanje informacije sta pravzaprav temelja grafičnega uporabniškega vmesnika, kakršnega poznamo danes. V tistem času sta drastično spremenila računalnike, ki so jih do tedaj razumeli kot suhoparna in linearna pomagala za pretvorbo števil.

Ko je leta 1955 doktoriral, se je Englebart zaposlil na Stanfordskem raziskovalnem inštitutu (SRI). Po številnih patentih, ki so sledili, je Englebart svoje ideje in raziskovalni načrt leta 1962 objavil v eseju "Razširjanje človekovega intelekta" (angl. Augmenting Human Intellect), leta.

"To je bil ogromen skok v razmišljanju za leto 1962. Edini računalniki, ki so obstajali v tistem času, so bile ogromne centralne enote (mainframes.) Tipična komunikacija med uporabniki in njimi se je odvijala skozi nekaj, kar se je imenovalo 'serijska obdelava podatkov'. Uporabnik je priložil program na serijah luknjanih kartic, računalnik je začel izvajati program v nekem določenem času, rezultati pa so prišli v roku od nekaj ur do nekaj dni. Sama ideja o uporabniku, ki vnaša ukaze na tekstualnem terminalu v realnem času (v tedanjem žargonu imenovanem 'time-sharing' oziroma delitev časa) je bila tedaj smatrana kot zelo radikalna." [1]

Esej je Douglasu zagotovil nujno potrebno financiranje za razvoj tehnologije, kar je privedlo do ustanovitve Centra za raziskovanje razširitev (ARC) na SRI in prve velike javne predstavitve pred več kot 1000 računalniškimi strokovnjaki leta 1968.

"Mati vseh predstavitev", kot je bila kasneje imenovana, je bila multimedijski spektakel. Spremljana je bila s kamerami, usmerjenimi v obraz, obe roki in mali zaslon pred njim (da bi lahko verodostojno spremljali vse te nove, do tedaj še nevidene koncepte). Douglas Englebart je predstavil NLS ("oN-Line System" oziroma "Vmreženi sistem"). Ime je dobil po zmožnosti, da se prek mreže poveže z drugimi računalniki.

Slika 1: NLS

Mali CRT zaslon dagonale 5" je prikazoval vektorsko grafiko, medtem ko je Douglas upravljal NLS s pomočjo treh upravljavskih naprav. Prva je bila standardna tipkovnica, druga je bila majhna specializirana, t. i. "akordična" tipkovnica s petimi tipkami, katerih kombinacije so omogočale 32 različnih vnosov (kar je bilo npr. dovolj za vse črke abecede), medtem ko je bila tretja, najzanimivejša naprava, podobna čudni majhni škatli s tremi gumbi na vrhu. Prek dolgega kabla je bila povezana z NLS-jem – to je bila prva miška. Engelbart jo je samostojno izumil, sodelavec Bill English pa mu jo je kasneje pomagal izboljšati. Prvi prototip miške je imel dve majhni kolesci, ki sta se kotalili po delovni površini in bili povezani z notranjimi potenciometri. Z izumom miške je prišel tudi kazalček, ki je na NLS-ju izgledal kot vodoravna in navpična puščica, usmerjena navzgor. Englebartova razvojna ekipa je kazalček imenovala "hrošč" (angl. bug), a se ta naziv ni dolgo obdržal.

Slika 2: Akordična tipkovnica (zgoraj in spodaj levo) in miška (spodaj desno)

V dolgi predstavitvi so bile predstavljene številne novosti. Med njimi so hypertext povezovanje, vsebinska pomoč, mrežno kolaboracijsko delo na dokumentih, e-pošta, instantno dopisovanje, večokenski način (okna so bila še brez okvirja, kar je bilo malce begajoče), nadzor različic dokumentov, celo video konferenca.

"Mnoge od prikazanih stvari na tem maratonskem predavanju so delovale, kot da prihajajo iz desetletij oddaljene prihodnosti in večina ljudi v občinstvu je imela težave z dojemanjem videnega." [1]

Vse prikazane inovacije sistema NLS so imele nalogo razširiti človeško znanstveno delovanje. Poudarek je bil skladno z Englebartovo vizijo na tem, da se omogoči uporabniku kar se le da močan nadzor, in ne na enostavnosti uporabe. Tako sta bila težavno učenje in privajanje na sistem glavna vzroka za padec NLS-ja v letih, ki so sledila. Uporabniki so potrebovali izkušnje v

programiranju v ozadju samega sistema, morali so si zapomniti zapleteno zaporedje ukaznih kod in ni bilo "point and click" vmesnika v pravem pomenu besede. Zaradi teh razlogov, poleg tega pa še zaradi pomanjkanja denarnih sredstev za nadaljnje raziskave, je odšlo veliko vrhunskih SRI raziskovalcev, med katerimi jih je večje število končalo v Xeroxevem Palo Alto raziskovalnem centru.

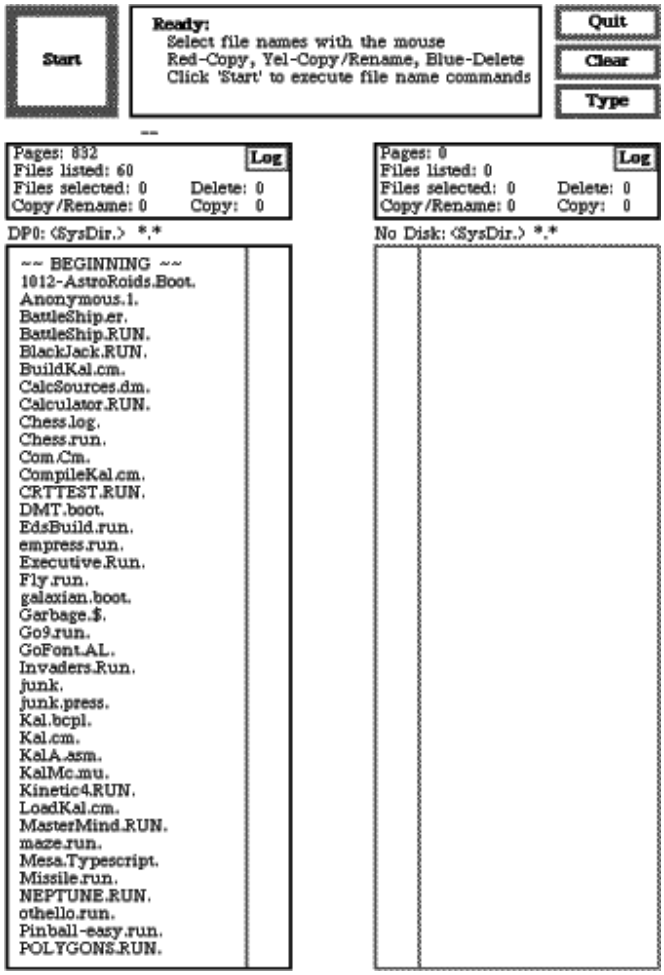
2.3 Raziskovalni center Palo Alto

Po Englebartovi veliki predstavitvi je ljudem postajalo vse bolj jasno, v katero smer se giblje računalniška tehnologija. Ni si bilo več težko zamisliti sveta, v katerem ljudje iz različnih krajev sveta istočasno delajo na skupnih projektih in si pri tem delijo elektronske dokumente, ki jih prikažejo in uredijo na računalniških zaslonih. Takšna vizija prihodnosti je dala misliti podjetju Xerox, ki je svoje bogastvo zgradilo s prodajo fotokopirnih naprav. V strahu pred "prihodnostjo brez papirja" se je Xerox takoj odločil trdno zavzeti položaj najmočnejšega igralca na takšnem tržišču. V ta namen je leta 1970 ustanovil raziskovalni center Palo Alto oziroma PARC.

PARC je obljubljal veliko raziskovalne svobode ter neomejeno širino pri razmišljanju in razvoju novih zamisli. To je privabilo najboljše računalniške strokovnjake v državi in povzročilo ustvarjalno in plodno delovno okolje. Eden od prvih izumov je bil laserski tiskalnik, ki je zahteval višji grafični način obdelave in priprave dokumentov za tiskanje. Ker takrat še ni obstajal računalnik, ki bi to lahko omogočil, je PARC leta 1973 razvil svoj računalnik. Xerox Alto je imel v tem časovnem obdobju miniaturne delovne dele sestavljene v ohišju, ki je ga je bilo mogoče postaviti pod delovno mizo. Najzanimivejši del je bil zaslon, postavljen v portretnem položaju velikosti lista A4. Imel je rastrsko grafiko ločljivosti 606x808 pik, piksli pa so bili lahko vključeni ali izključeni posamezno. To je bila bistvena razlika v do tedaj tipičnih terminalih, ki so imeli fiksne črke ali vektorsko grafiko NLS-ja, ki je lahko prikazovala samo besedilo in ravne črte. Alto se je upravljalo s pomočjo tipkovnice in modernizirane različice Englebartove miške s tremi gumbi.

Slika 3: Xerox Alto

Prva programska oprema, napisana za Alto, je bila grafično še zelo primitivna in nedodelana. "Okna", kakršna poznamo danes, še niso obstajala, logično ločljive celote vmesnika so bile ločene samo z izrisanimi okvirji, kar vidimo na primeru upravljalnika datotek (slika 4). Aplikacije niso imele uniformiranih grafičnih vmesnikov, zato je takšna pomanjkljivost nedefiniranega ogrodja za njihovo izdelavo programerjem novih aplikacij predstavljala dodatno skrb pa tudi napor za uporabnika, ki je zahteval dodatno učenje in navajanje na vsako kontrolno shemo nove aplikacije. PARC-ovi znanstveniki so spoznali, da je treba izdelati programska in konceptualna ogrodja za razvoj vmesnika, in tako je nastal Smalltalk, prvi moderni GUV.



Slika 4: Altov upravljalnik datotek

2.4 Smalltalk

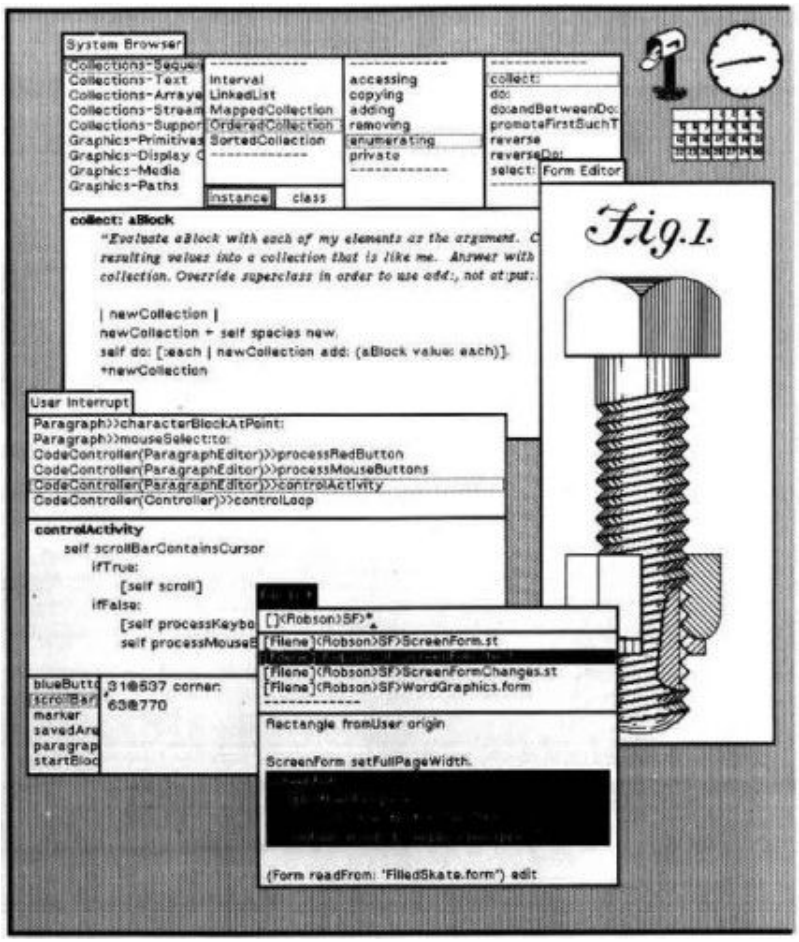
"Smalltalk je bil spočet kot programski jezik, njegovo razvojno okolje je tako enostavno za uporabo, da ga je lako razumel tudi otrok." [1]

Ta prvi pravi objektno usmerjeni programski jezik, ki je črpal navdih iz Simula, se je pojavil leta 1972 kot rezultat dela skupine raziskovalcev v PARC-u, ki jih je vodil Alan Kay. Imel je avtomatično upravljanje pomnilnika (kot Java), programska koda in podatki pa so bili vsebovani v enotah pod imenom objekti. Objekte iz enega programa so lahko uporabljali tudi drugi programi, neodvisno od njihove prvotne implementacije. Prva javno dostopna različica je bila Smalltalk-80. Poleg tega, da je bil programski jezik in razvojno okolje za grafični vmesnik, je bil Smalltalk tudi okolje za zagon lastnih programov. Pri zagonu na Altu je Smalltalk prevzemal vse upravljanje in prikaz grafičnega vmesnika.

Največji pomen tega grafičnega okolja je v tem, da so v njem prvič predstavljene številne nove izboljšave, objekti in značilnosti grafičnega vmesnika, ki so vse do današnjih dni postali paradigma vseh novih uporabniških grafičnih vmesnikov, nastalih po njem. Okna so se prvič pojavila v okvirjih, tako da se jih je lahko razlikovalo med seboj, vsako okno je imelo na vrhu naslovni trak, s pomočjo katerega se je lahko prepoznalo vsebino okna in ga premaknilo po zaslonu (slika 5). Okna so se med seboj lahko prekrivala in pojavila se je potreba po razvrščanju oken. Rešitev je bila v kaskadnem razvrščanju oken po zaslonu z ustrezno funkcionalnostjo. Ta funkcionalnost pomeni, da s klikom na posamezno okno tega pripeljemo na vrh sklada in dosežemo, da je popolnoma pregleden. Okna so vsebovala tudi novosti, kot so drsni trakovi, radijski gumbi in dialog. Tudi pojavni (angl. pop-up) meniji so prvič prišli na svet. Pojavljali so se pri kliku gumba na miški, in to na trenutni poziciji kazalčka na zaslonu. Imeli so hierarhično vsebino možnosti, ki so se posebej navezovale na del vmesnika, za katerega je bil ta meni poklican (del pod kazalčkom). Koncept ikon je tudi nekaj, kar nam je prinesel Smalltalk. Ikone kot miniaturne reprezentacije programov in dokumentov so služile za lažje vizualno razpoznavanje, zagon in upravljanje.

Močna kombinacija Altove strojne opreme in Smalltalka kot grafičnega okolja je ponujala osebni računalnik za polno pisarniško uporabo pri ustvarjanju in tiskanju dokumentov, mrežnem kolaboracijskem delu in komunikaciji po elektronski pošti. Ta delovna enota je pravzaprav najpomembnejši prednik današnjih PC-jev in osnovni načrt, po katerem so ustvarjeni tudi današnji "moderni" uporabniški grafični vmesniki na PC-ju.

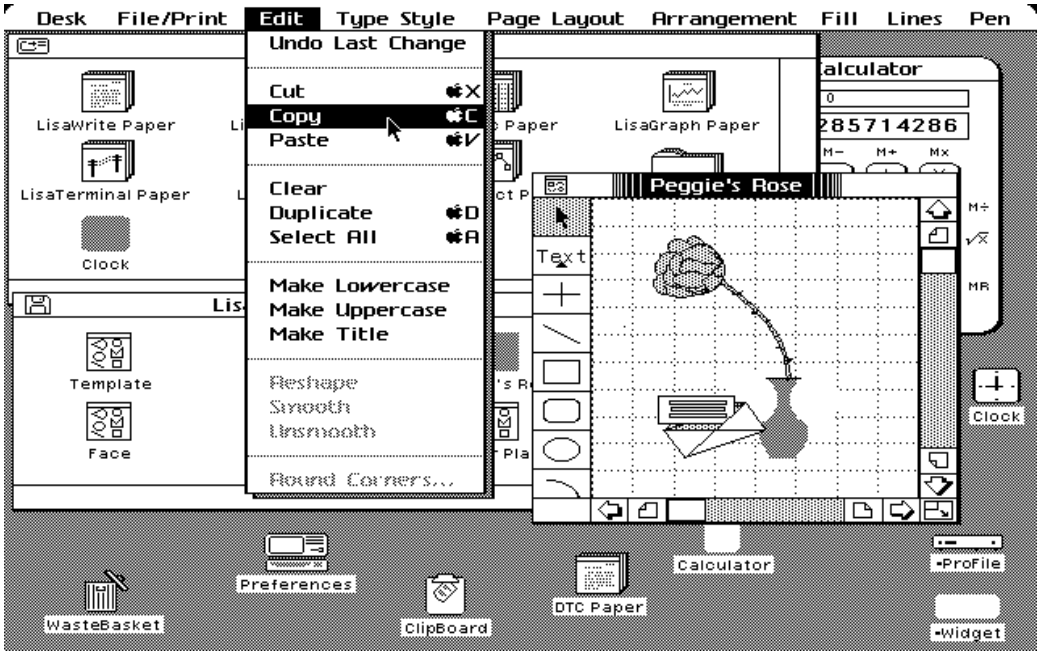
Altova usoda ni bila tako svetla, kot bi lahko sklepali. Čeprav so mnogi PARC-ovi raziskovalci želeli, da matični Xerox komercializira novi in znatno cenejši Alto III, je upravni odbor Xeroxa to zavrnil. Malo kasneje, leta 1981, je Xerox vseeeno predstavil komercialno osiromašeno različico Alta z imenom Xerox Star 8010 Document Processor, vendar je bilo že prepozno, saj so najboljši raziskovalci in strokovnjaki že zapustili Xerox in odšli v druga podjetja.



Slika 5: Razvojno in grafično okolje Smalltalk

2.5 Apple

To podjetje z zanimivo preteklostjo je nastalo leta 1976 v garaži. Ustanovili so ga Steve Wozniak, Steve Jobs in Ronald Wayne. Za svoje napredne začetke in izvrstno konkurenčnost na računalniških sejmih med tekmeci, kot sta bila Tandy in Commodore, se Apple lahko zahvali Wozniakovi genialnosti, ker je prav on z lastnimi rokami zlagal prve računalnike v prej omenjeni garaži, in prav njegovo znanje je tisto, ki je Apple Computers takoj označilo za napredno podjetje, kljub skromnim začetkom brez denarne in tržne moči. Tržno moč je kasneje priskrbel Steve Jobs s svojim poslovnim nosom in čutom za posel, s čimer je zagotovil velike prihodke od vlagateljev in poslovnih zvez, ki so podjetje lansirale v resne poslovne vode. Na samem začetku je bil Apple II, ki ga je oblikoval in zgradil Wozniak, to, kar jim je prineslo veliko pozornosti in močne vlagatelje. Predstavljen leta 1977 je privabil nove zaposlene in odprl Apple proizvodno linijo. Apple III, ki je izšel leta 1980, je dosegel komercialni neuspeh, kljub temu pa se je Jobs gibal v pravo smer. Leta 1979 je obiskal Xeroxov PARC in se spoznal z GUV-om. Takoj je dojel, da bodo prihodnji računalniki uporabljali prav takšne grafične vmesnike, zato je začel razvijati GUV za naslednji Applov projekt, Liso. K temu so drastično pripomogli novi PARC-ovi inženirji, ki so bili nezadovoljni z usodo Alta. Upali so, da bodo nadaljevali prejšnje delo, vendar tokrat na nečem, kar bo končno poslano na tržišče in bo hitro postalo zelo popularno. Lisin vmesnik je bil narejen z nadaljnjim oblikovanjem Smalltalkovega koncepta. Razvit je bil prvi univerzalni trak s spustnimi meniji na vrhu zaslona. Trak je vseboval vse pomembne menije za trenutno fokusirano aplikacijo. Meniji so imeli zraven izbranih možnosti kljukice, nedostopne pa so bile zasenčene (slika 6). Drsne trakove proporcionalne velikosti so zamenjali s fiksno velikimi. Dodan je bil tudi novi koncept bližnjic na tipkovnici za pogosto uporabljene možnosti v menijih. Inženirji iz PARC-a so prinesli s seboj tudi miško. Miška s Xerox Starja je imela dve tipki, tista napravljena za Liso pa je "poenostavljena" na samo eno tipko, kar je povzročilo ustvarjanje novega koncepta "dvojnega klika". Prvi klik je bil za označevanje, naslednji pa za zagon aplikacije oziroma odpiranje dokumenta. Vsak dokument se je odprl v ustrezni aplikaciji, kar so omogočili posebej za to implementirani datotečni razredi. Lisa je bil prvi računalnik, na katerem so bile, razen samih aplikacij in dokumentov, vse datoteke v sistemu označene z ikonami in prikazane v hierarhični strukturi. Dodana je bila tudi ikona koša za smeti, ki je vsebovala zbirko dokumentov, pripravljenih za brisanje. V koš za smeti se je kot še na nekaterih drugih mestih lahko povleklo več datotek, zahvaljujoč novi "odvleci-in-izpusti" možnosti, ki je dopuščala premeščanje ali kopiranje skupine (predhodno označenih) datotek na drugo lokacijo.



Slika 6: Vmesnik Apple Lise

Zaradi vseh teh novih konceptov je bila Apple Lisa leta 1983 glavni, če že ne tudi končni odmik v GUV-u od začetnega na NLS-ju in oblikovanega na Xerox Altoju proti nečemu, kar bi lahko imenovali glavni načrt modernega GUV-a na osebnih računalnikih. Po Lisinem vmesniku se stvari niso bistveno spreminjale, tako da imamo danes zelo podobne, skoraj enake grafične vmesnike na modernih PC operacijskih sistemih.

2.6 Širjenje GUV revolucije

Apple je leta 1984 dal na tržišče Macintosh, ki je zahvaljujoč nizki ceni preživel predrago Liso in nadaljeval tradicijo podjetja. V tem času so vsi proizvajalci, ki so hoteli del tržišča, začeli predstavljati svoje GUV rešitve. VisiOn, Tandy DeskMate, GEM, Amiga Workbench, GEOS, IBM OS/2 so le nekatere od njih. Eni so kopirali Apple, drugi pa so imeli svoje lastne, vendar tehnološko zastarele in nerodne vmesnike. Našli so se tudi takšni, ki so uvedli manjše spremembe. To sicer niso bile kritične inovacije, ki bi spremenile načrt Lise do neprepoznavnosti, nasprotno, ta je ostal osnova vse do danes.

Idejo odlagališča (angl. dock) na dnu zaslona je leta 1987 prvi predstavil Acorn Computers iz Velike Britanije na svojem računalniku A305/A310, ki je bil hkrati tudi prvi 32-bitni računalnik z arhitekturo RISC. Odlagališče je bilo narejeno kot trak na dnu zaslona, ki je vseboval bližnjice za zagon programov in sistemskih funkcij.

Steve Jobs je leta 1985 zapustil Apple in se ponovno vrnil leta 1996. V obdobju, ko ni bil v Applu, se je lotil svojega velikega projekta, ki je pripeljal do računalnika NeXT in njegovega operacijskega sistema NeXTSTEP. V njem sta prvič predstavljena dva dodatka v naslovnem traku okna, ki ju vsi poznamo še danes: "X" – simbol za zapiranje okna v zgornjem desnem kotu – in vertikalni spustni meni v zgornjem levem kotu.

2.6.1 X Window System

Izumljen je bil leta 1984 na MIT-u in je osnova za razne izvedbe GUV-a, istočasno pa tudi mrežni protokol, ki omogoča skupni grafični prikaz na vmreženih računalnikih. Je ogrodje za izrisovanje in premikanje oken pa tudi za upravljanje s pomočjo miške in tipkovnice. Kot takšen ne predstavlja uporabnega GUV-a, temveč služi le kot nosilec za t. i. "upravitelje okna" (angl. Window managers), ki skrbijo za pojavljanje in odstranjevanje oken ter upravljanje gradnikov. Na upravitelja okna se postavi dodaten sloj, t. i. "namizno okolje" (angl. Desktop environment), ki omogoča končno uporabnost in občutek izpopolnjenega GUV-a. X je torej strojno-abstraktni sloj, postavljen v jedro operacijskega sistema, nanj pa lahko postavimo popoln in ločen GUV (upravitelj okna in namizno okolje), katerega izvedba je odvisna izključno od domišljije programerja, ki ga gradi. To je pomembno zato, ker zelo razširi populacijo potencialnih programerjev GUV-a, ki se naslanjajo na X. Če nekdo želi ustvariti popoln GUV za X, mu ni treba imeti znanja za izgradnjo lastnega računalnika, kot je Apple, niti mu ni treba skrbeti za strojno opremo kateregakoli računalnika ali razvijati svojega operacijskega sistema. Namesto tega lahko preprosto izkoristi ves svoj čas za razvijanje novih ali izboljševanje obstoječih konceptov GUV-a, torej se posveti izključno tehnikam interakcije uporabnika z računalnikom. Kasneje bomo videli, da tudi to do danes ni bilo dovolj, da bi prišlo do radikalnih preobratov pri načinu upravljanja GUV-ov, kot je na primer tisti z gestami.

Pred koncem 80. let prejšnjega stoletja so se začeli pojavljati GUV-i za X, ki so se zagnali na UNIX delovnih postajah proizvajalcev, kot so AT&T, Sun in HP. Ti vmesniki so, čeprav so bili kopija koncepta Windows na vrhu močne lupine UNIX, uvedli tudi svoje izboljšave. Okno, na

katerega je uporabnik premaknil kazalček, se je aktiviralo, ne da bi ga bilo treba predhodno označiti. Tako se je lahko tipkalo v aktiviranem oknu, ne da bi ga odvlekli na vrh sklada vseh oken. Na žalost se je s tem izgubila preglednost.

2.6.2 Microsoft

Po uspehu z operacijskim sistemom MS-DOS se je Microsoft pri skupnem razvoju sistema OS/2 z IBM-om odločil vriniti v GUV vode. Leta 1985 je proizvedel grafično razširitev za MS-DOS, Microsoft Windows. Microsoft je že prej prihajal v stik z Applovim GUV-om kot zunanji proizvajalec programske opreme za njihov Macintosh. Macintoshev bogat GUV, ki je nasledil tistega z Lise, je najverjetneje vplival na Microsoft, tako da je prva verzija Windowsa že imela drsne trake, gradnike za upravljanje oken, menije in ikone. Edina specifična in zelo prepoznavna sprememba, ki jo je vnesel Microsoft, je bila ta, da je imelo vsako okno svoj zaseben menijski trak takoj pod naslovnim trakom. Zaradi dileme glede preglednosti so okna v prvi različici zavzemala mesta drug drugega na delovni površini in bila postavljena mozaično. Vendar se je v Windows 2.0 leta 1987 Microsoft odločil za preklapljajoča okna in tako je ostalo vse do danes. V začetku 90. je z različicami 3.0 in 3.1 Windows dosegel ogromno popularnost, čeprav je imel še vedno manjšo funkcionalnost kot Applov GUV. Leta 1995 se je pojavil Windows 95 in učvrstil Microsoft na vodečo pozicijo v prodaji GUV-ov ter postal najuspešnejši operacijski sistem do takrat. Najprepoznavnejša sprememba v GUV-u, ki jo je prinesel s seboj, je bil delovni trak z gumbom "Start" in pripadajočim menijem. Start meni je vseboval ikone nameščenih aplikacij in tako označeval logični začetek uporabnikovega dela. Sam delovni trak je vključeval dinamične gumbe aplikacij, ki se trenutno izvajajo. Z zagonom poljubne aplikacije se njej pripadajoči gumb pojavi na delovnem traku. S klikom nanj lahko skrčimo in povečamo aplikacijo oziroma jo postavimo v ospredje na delovni površini, tako da se prestavi na vrh sklada oken. To je omogočilo hitro in enostavno preklapljanje med odprtimi okni. Na desni strani traka je bil sistemski pladenj, tj. majhen prostor z uro in številnimi ikonami, ki so prikazovale procese, ki tečejo v ozadju brez odprtega okna na delovni površini.

Delovni trak je največji Windowsov prispevek GUV-u in hkrati tudi zadnji pomemben. Čeprav je v naslednjih izdajah Windowsa (Vista in 7) do danes predstavljen nov GUV – Windows Aero – se je večina novosti gibala od grafičnih efektov, kot je prozornost, do t. i. bonbončkov (angl. Eye

candy), kot so animacije, lepše in animirane ikone itd. Pomembnejših sprememb v načinu uporabe GUV-a, ki bi pospešile, poenostavile in obogatile delo, po našem mnenju ni.

Windowsov GUV je zaradi popularnosti svojega prevladujočega OS-a danes najbolj znan in razširjen v svetu osebnih računalnikov. Kot takšen se je vsilil kot svojevrsten standard uporabnikom PC-ja po vsem svetu. Njihov morebitni prehod na alternativni OS bi prinesel samo dodatno breme in probleme zaradi težav pri navajanju na drugačen GUV. Zato so mnogi GUV-i začeli oponašati Windowsov model, da bi čim bolj posnemali Windowsove izkušnje pri uporabnikih. Nekateri od takšnih GUV-ov so zgrajeni na zelo močnih operacijskih sistemih, katerih zmožnosti praktično predstavljajo plodna tla za razvoj alternativnih in kakovostnejših načinov upravljanja, vendar so ti potenciali do danes ostali neizkoriščeni. Eden od takšnih je GNU/Linux.

3 ZAKAJ GESTE Z MIŠKO?

Pred več kot 43 leti je Douglas Englebart dojel, da je treba način naše interakcije z računalnikom spreminjati, dopolnjevati in izboljševati. Njegova zapuščina je utrdila mnenje, da je kakovost človeškega stika in komunikacije s strojem zelo pomemben del računalniške prihodnosti. Prihodnost je bila ustvarjena in po petnajstih letih smo prvič dobili računalnik s popolnim in uporabnim grafičnim uporabniškim vmesnikom, ki je postal temelj za vse bodoče različice. Takrat se je osebni računalnik s svojim intuitivnim in učinkovitim GUV-om predstavil kot resno nadomestilo za tradicionalno opravljanje pisarniškega dela. Stvari se niso ustavile pri tem in osebni računalnik je do danes zrasel v orodje za pisno, slušno in vizualno sporazumevanje, urejanje in izpisovanje dokumentov, obveščanje, informiranje, igranje, učenje itn. Nahaja se znotraj skoraj vseh območij človeškega delovanja in tudi za navadnega človeka je resnično del vsakdanjega življenja, vendar tega statusa ne bi dosegel brez GUV-a, ki je navadnemu, neračunalniškemu tipu človeka razumljiv in enostaven za uporabo.

Osebni računalniki so se od prve uspešne komercialne uporabe modernega GUV-a močno razvili. Od pojavitve Apple Lise leta 1983 do danes je računalnik napredoval skozi številne spremembe. Močno sta se povečali hranitvena kapaciteta in procesorska moč, razširil se je pomnilniški prostor, okorne katodne zaslone so zamenjali tanki LCD-ji. Vzporedno s strojno se je pospešila in nadgradila tudi programska oprema. Uporabnik je dobil na voljo veliko novih programskih izbir in rešitev, veliko naprednejše in vsestranske aplikacije ter boljša in hitrejša orodja. Edino, kar se ni spremenilo veliko, morda celo nič, je način upravljanja z računalnikom. GUV je doživel veliko estetskih sprememb in dodatkov, ampak nobenega resnega odmika od modela WIMP in standardnega načina upravljanja operacijskega sistema z miško in tipkovnico. Model GUV-a je ostal enak kot pred 30 leti, ko je služil kot paradigma delovnemu pisarniškemu prostoru z njegovimi datotekami, mapami in namizji. Razlogi za zastoj v razvoju ležijo v poti najmanjšega odpora, ki ji sledijo razvijalci. Dosedanji GUV je potrjen model uspeha na trgu, zaradi njega je osebni računalnik dosegel ogromno priljubljenost med ljudmi in vsaka večja konceptualna sprememba pomeni tržno tveganje. Velika večina ljudi je navajena na dosedanje načine upravljanja. Vsaka nova sprememba predstavlja za uporabnike obveznost novega učenja in prilagajanja ter izgubo časa, kar spet odvrne tiste, ki so privajeni na stari sistem. Še en razlog leži

v pomanjkanju tehnične podpore, ki otežuje razvoj vmesnikov, ki niso zasnovani na modelu WIMP [2].

Navedeni razlogi se zdijo popolnoma upravičeni in zadostni za obdržanje starih konceptov GUV-a in dosedanjega modela upravljanja. Vendar si moramo zastaviti vprašanje: ali bo vedno tako? Vsak nov dan prinaša tehnološke izboljšave. Tehnologija postaja naprednejša, močnejša in vsestranska. Nove možnosti sistemov prinašajo nove izbire v GUV-u. Pojavljajo se nove povezave, novi gumbi in nove vrste podob v aplikacijah. Uporabniki, ki razumemo razliko med navadnim WIMP delom vmesnika in novimi dodatki, se lahko znajdemo v taki zmešnjavi. Problem doleti uporabnika začetnika, ki je od prvega trenutka na računalniku obremenjen s preveliko količino informacij. V današnjem morju elektronskih pripomočkov, smo se prisiljeni učiti posameznih vmesniških jezikov oziroma protokolov za komunikacijo z vsakim izmed njih, naj bo to kalkulator, DVD-predvajalnik ali operacijski sistem osebnega računalnika. Kognitivna obremenitev, o kateri piše Buxton [3], je na začetku učenja takšnega HCI protokola zelo povečana in otežuje normalno delo. Problem lahko ponazorimo s primerom avtomobila.

Voznik začetnik, ki se je šele seznanil z vmesnikom avtomobila, bo imel težave v prometu. Protokol za upravljanje avtomobila vsebuje pravočasno zaviranje, pospeševanje pri odhodu, prestavljanje v pravem trenutku, gibanje nazaj z uporabo retrovizorja itn. Protokol komunikacije v prometu pa zahteva nove podrobnosti: ustrezno interpretacijo signalnih luči, razumevanje cele množice prometnih znakov, osredotočenost na pešce, upoštevanje prednosti pri zavijanju itn. Napreden voznik se lahko osredotoči na promet, ker avtomobil upravlja avtomatično in mu o tem sploh ni treba več razmišljati. Mlademu vozniku, ki želi prispeti na cilj, protokol za upravljanje avtomobila predstavlja kognitivno obremenitev, ker nanj ni še privajen. Takšna vrsta obremenjenosti mu preprečuje, da se skoncentrira na osnovno nalogo – skrb za prometne razmere.

Podobno kot mladi voznik pri avtu neizkušeni uporabnik računalnika naleti na težave z uporabo GUV-a. Odpiranje, zapiranje, krčenje, večanje in premikanje okna, uporaba navadnega in dvojnega klika na različnih delih vmesnika, kopiranje in lepljenje datotek, sprehajanje po datotečnem sistemu ... Vse to je del komunikacije HCI, ki novemu uporabniku predstavlja kognitivno obremenitev. Namesto hitrega opravljanja enostavne glavne naloge, kot je izpis dokumenta ali pošiljanje elektronske pošte, se mora neizkušeni uporabnik ukvarjati z učenjem protokola komunikacije z računalnikom. Mora vedeti, kako so videti posamezni gradniki GUV-a, ki opravljajo določene naloge, mora jih znati najti v množici podob. Obvladovati mora

sestavljene naloge in razumeti sestavne dele vhoda [3]. To mu otežuje in upočasnjuje delo ter ga na koncu odbije od uporabe računalnika. Na tak način računalnik izgubi vlogo orodja in pripomočka ter postane nov problem in nova obremenitev. Namesto da olajša opravljanje naloge uporabniku, sam postane naloga.

"...digitalni računalniki bi lahko preskrbeli najhitrejšo metodo za povečanje možnosti človeka, da se loti zapletenega problemskega stanja, za pridobitev razumevanja, ki ustreza njegovim posebnim potrebam in za izpeljavo rešitve problemov." [4]

Da bi lahko pospešili delo v zapletenih situacijah in omogočili dobro razumevanje postopka za hitro prihajanje do rešitve, računalnik potrebuje takšen način upravljanja, ki v samem začetku in kolikor je to možno razbremenjuje uporabnika, mu prizanaša z učenjem novega jezika in protokola komunikacije ter mu tako dovoli, da se maksimalno osredotoči na osnovno nalogo. Rešitev tega problema se začne z uporabo enostavnega vizualnega jezika komunikacije HCI: gest.

Geste so človeku prirojen način komunikacije. Raziskave v kognitivni psihologiji so pokazale, da uporaba gest igra blagodejno vlogo v spominu, komunikaciji, učenju in reševanju problemov [5]. Človeški delovni spomin je podoben računalniškemu pomnilniku. To je zmožnost pomnjenja informacije za kratek čas (nekaj sekund) in tisto, kar nam omogoča, da ostanemo osredotočeni na osnovno nalogo. Rezultati raziskav so dokazali, da uporaba gest kot pomoči v razmišljanju in vsakdanjem govoru zmanjšuje kognitivno obremenitev delovnega spomina na minimalno stopnjo in ga tako sprošča pri drugih nalogah. Učinkovitejše razmišljanje, učenje in na splošno boljši rezultati so možni, ker se kakovostno osredotočamo na nalogo in izbiro pravilne strategije. Geste z rokami uporabljajo otroci, ko se učijo komunicirati. Tudi veliko odraslih ljudi v vsakdanjem govoru uporablja geste. Z gestami smo rojeni. To je edini način komunikacije, ki od samega začetka uporabe zmanjšuje kognitivno obremenitev. Ko pride do zastoja v govoru tujega jezika, ker se ne moremo spomniti določene besede, geste lahko pomagajo nadaljevati komunikacijo. Vse navedeno je dokaz, da so geste skupen jezik, ki ga razumejo vsi ljudje, in nekakšna vrsta kognitivnega protokola za obravnavanje in komunikacijo z osebno okolico. Kot take predstavljajo idealno dopolnitev oziroma celo nadomestek za dosedanje HCI načine upravljanja.

Po J. Raskinu [6] se domačnost dizajna uporabniškega vmesnika oziroma seznanjenost ljudi z njim enači z intuitivnostjo. To je ovira pri ustvarjanju boljših dizajnov vmesnika, saj se udomačeni in pravi intuitivni dizajni izenačujejo po vrednosti, intuitivni pa se na koncu zavržejo zaradi neraziskovanosti pristopa in težje implementacije.

Geste, kakršne bomo obravnavali, niso domače v GUV-u, so pa resnično intuitivna oblika dizajna. Težave implementacije v našem primeru bodo odpadle, ker bomo uporabljali že dostopne in brezplačne programe kot enostavno nadgradnjo obstoječega GUV-a. Visoka cena potrebne strojne opreme (kot npr. pri rešitvah, ki uporabljajo specializirano opremo za sledenje kretnjam telesa) prav tako ne bodo problem, saj bodo naše geste ustvarjene s popolnoma osnovno in poceni opremo: miško.

4 METACITY

V GNU/Linux svetu osebnih računalnikov je trenutno najbolj razširjena Ubuntu distribucija [7]. Njeno privzeto namizno okolje, ki omogoča GUV, je GNOME. Zato lahko rečemo, da je GNOME najbolj razširjen GUV operacijskega sistema GNU/Linux. GNOME je enostavno in uporabniku prijazno namizno okolje [8], kar je nedvomno vplivalo na njegovo priljubljenost. Uporabniki so dobro seznanjeni z njegovim načrtom GUV-a. V nasprotju z nekaterimi drugimi namiznimi okolji GNOME nima svojega upravljalnika oken in je pri tem odvisen od tretje strani. Čeprav je njegova nova različica (3.0) v trenutku nastajanja tega diplomskega dela privzela nov upravljalnik oken, je GNOME svojo široko bazo uporabnikov in priljubljenost [9] pridobil z upravljalnikom oken Metacity.

4.1 Prednosti upravljalnika oken Metacity

Metacity je upravljalnik oken, ki sledi predvsem enostavnosti in uporabnosti. Ne trudi se z nepotrebno funkcionalnostjo, tako da ima zgolj pričakovane osnovne sposobnosti, kot so odpiranje, zapiranje, premikanje, krčenje, večanje, razpiranje, spreminjanje velikosti, preklapljanje med okni in risanje dekoracij oziroma naslovne vrstice in robov okna. Kljub prej naštetim osnovnim zmožnostim ponuja uporabniku pri upravljanju oken veliko več kot Windowsov upravljalnik. Nekatere od dodatnih možnosti so:

- premikanje s prijemom na katerokoli območje okna (poleg naslovne vrstice),
- spreminjanje velikosti s prijemom na katerokoli območje okna (poleg robov),
- avtomatični premik na robove, v oglišča in središče namizja,
- premik na sosednja navidezna namizja,
- premik na poljubno navidezno namizje,
- istočasni prikaz na vseh navideznih namizjih,

- spuščanje pod druga okna (na dno sklada) in dvig na vrh,
- horizontalno in vertikalno razpiranje,
- postavljanje v “zasenčeno” (angl. shaded) stanje.

Najpomembnejša razlika med Metacityjem in Windowsovim upravljalnikom je v tem, da zgoraj navedene osnovne in tudi dodatne funkcije v Metacityju lahko uporabljamo s pomočjo nastavljivih bližnjic na tipkovnici. Upravljalnik oken sistema Windows XP omogoča uporabo bližnjic za zgolj peščico osnovnih funkcij, na primer skritje vseh oken, prikaz vseh oken in razpiranje posameznega okna. Za vse drugo, kot je premikanje, skrčitev, povečanje posameznega okna in spreminjanje njegove velikosti, je treba uporabiti miško in klikniti na ustrezen gumb v območju dekoracije (naslovne vrstice in okvira) okna. Ta pomanjkljivost Windowsovega upravljalnika oken precej omeji razširljivost GUV-a in uporabniku onemogoči uporabo tipkovnice za njegovo hitro upravljanje. Prav tako ne podpira razširitev z orodji, ki bi omogočila celostni nadzor nad okni le z miško, kot je to mogoče storiti z uporabo gest. Treba je omeniti, da so spremembe, ki so prišle z novim GUV-om operacijskega sistema Windows 7, prinesle nekoliko razširjene možnosti upravljanja oken, a je teh premalo. Med novimi možnostmi upravljanja oken so premik okna na levo ali desno stran namizja ter skrčitev in povečanje okna z uporabo bližnjic na tipkovnici. A teh bližnjic, enako kot pri Windows XP, še vedno ni mogoče določiti ročno. Nespremenljivost bližnjic za pomembne funkcije predstavlja še eno ključno oviro pri poenostavljanju GUV-a in pospešitvi uporabnikovega dela. Če bi se bližnjice lahko spremenilo na uporabniku dostopen in prijazen način, bi jih bilo mogoče organizirati tako, da bi jih lahko priklical z eno roko brez spuščanja pogleda z zaslona na tipkovnico. To pa je svojevrstna oživitev koncepta akordične tipkovnice, ki jo je predstavil Englebart na “Materi vseh predstavitev”. Kljub Windowsu 7 in njegovim majhnim izboljšavam bomo v nadaljevanju raziskovali možnosti upravljanja v GUV-u Windows XP-ja ter izpostavljali razlike med GNU/Linuxom in Windows XP-jem, ki je v trenutku pisanja tega diplomskega dela še vedno najbolj razširjen [7]. Tudi če bi obravnavali izboljšave v GUV-u Windowsa 7, lahko trdimo, da so GUV-i GNU/Linuxa postavili veliko bogatejši nabor upravljanja oken, in to bistveno prej, kot je izšel Windows 7.

4.2 Sledenje konceptu akordične tipkovnice

Še preden začnemo spreminjati lastnosti Metacityja, je treba postaviti pogoje za izbiro bližnjic za posamezne funkcije. Ti pogoji nam bodo, kot navadnemu uporabniku, olajšali izbiranje novih bližnjic v prihodnosti in pomnjenje obstoječih. Če hočemo doseči prednosti akordične tipkovnice pri izbiri bližnjic, je treba paziti, da se približajo naslednjim pogojem:

1. da so v dovolj majhnem območju ter ne ovirajo dela in so lahko dosegljive levi roki brez spremembe njenega položaja,
2. da bo možnost, da jih najdemo brez gledanja, pri prvem dotiku roke s tipkovnico čim večja,
3. da so izbrane s tako logiko, da se lahko pomnijo vizualno ali tekstualno in da jih je enostavno najti, tudi kadar jih ne poznamo.

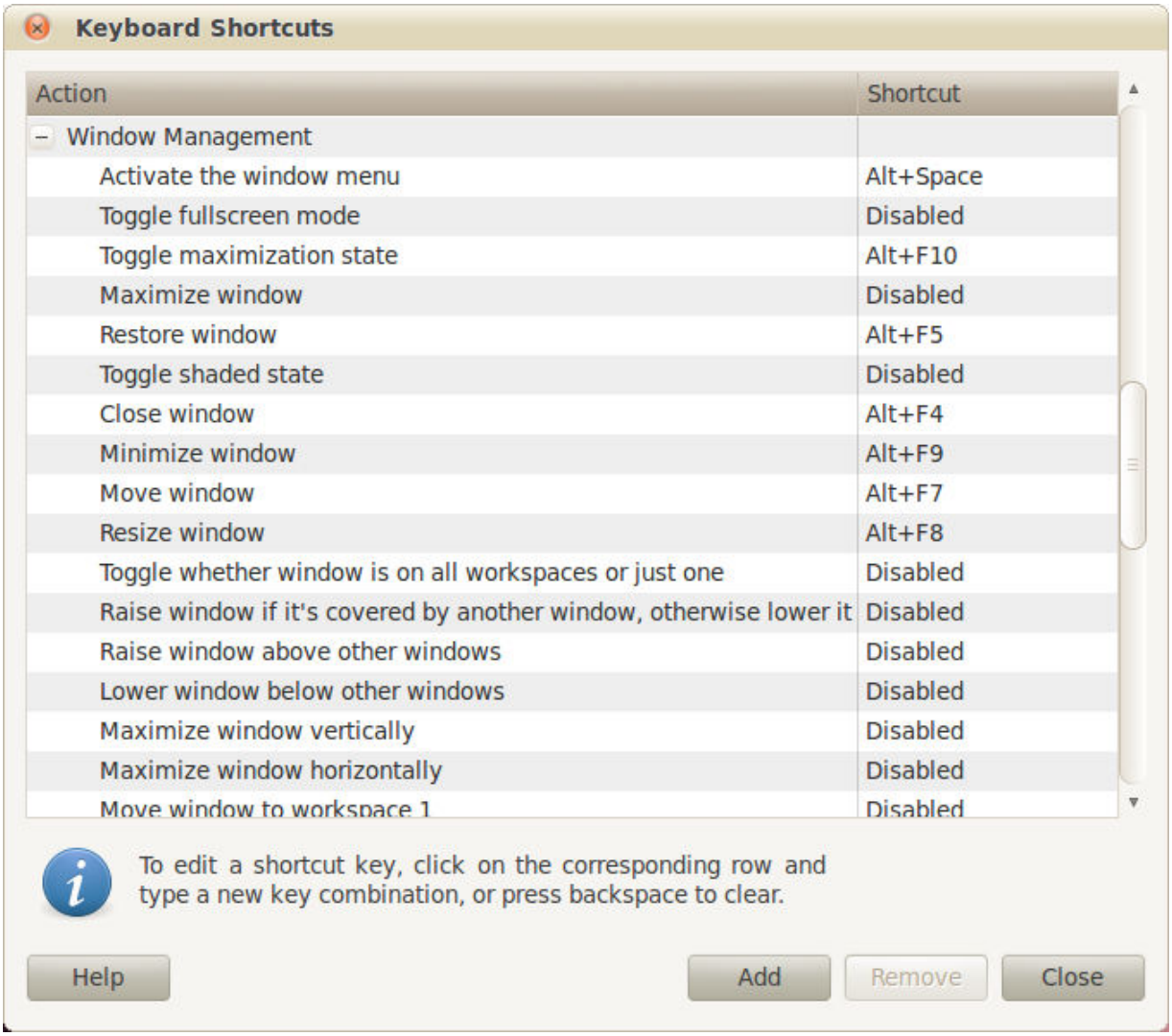
Prvi pogoj preprečuje postavljanje roke v čudne in neprijetne položaje in možne poškodbe pri delu ter omogoča hitrejšo in natančnejšo uporabo bližnjic. Prav tako pomaga pri izpolnitvi drugega pogoja. Cilj je najti optimalen, sproščen položaj roke, iz katerega so dosegljive vse pomembne bližnjice brez potrebe po večjem premikanju.

Drugi pogoj je najpomembnejši za tekoče in nemoteno delo. Sled uporabe bližnjic na tipkovnici se prekine vsakič, ko premaknemo roko iz prvotnega položaja, na primer, ko začnemo tipkati besedilo. Pri ponovni uporabi bližnjic je pomembno, da naša roka čim prej spet najde tisti optimalni položaj iz prvega pogoja. To naj se dogaja brez da bi gledali v tipkovnico ter s tem izgubljali čas in osredotočenost na delo. Ta problem lahko rešimo tako, da so bližnjice recimo v bližini spodnjega levega dela tipkovnice, ki ga lahko najdemo tudi “na slepo”. Ko postavimo roko v pravi položaj, je pomembno, da nam pri vsakem naslednjem iskanju bližnjic ni treba ponovno pogledati na tipkovnico. Najpogosteje uporabljane bližnjice je torej dobro izbrati tako, da so vedno pod prsti roke v optimalnem položaju. Njihov položaj naj služi kot orientacijska os, vse druge bližnjice pa naj bodo v neposredni bližini. Tako bo vsakdo lahko hitro našel in si zapomnil bližnjice, saj bo vedel, kje jih iskati, tako kot izkušen voznik ve, kje sta menjalnik in stopalka za plin.

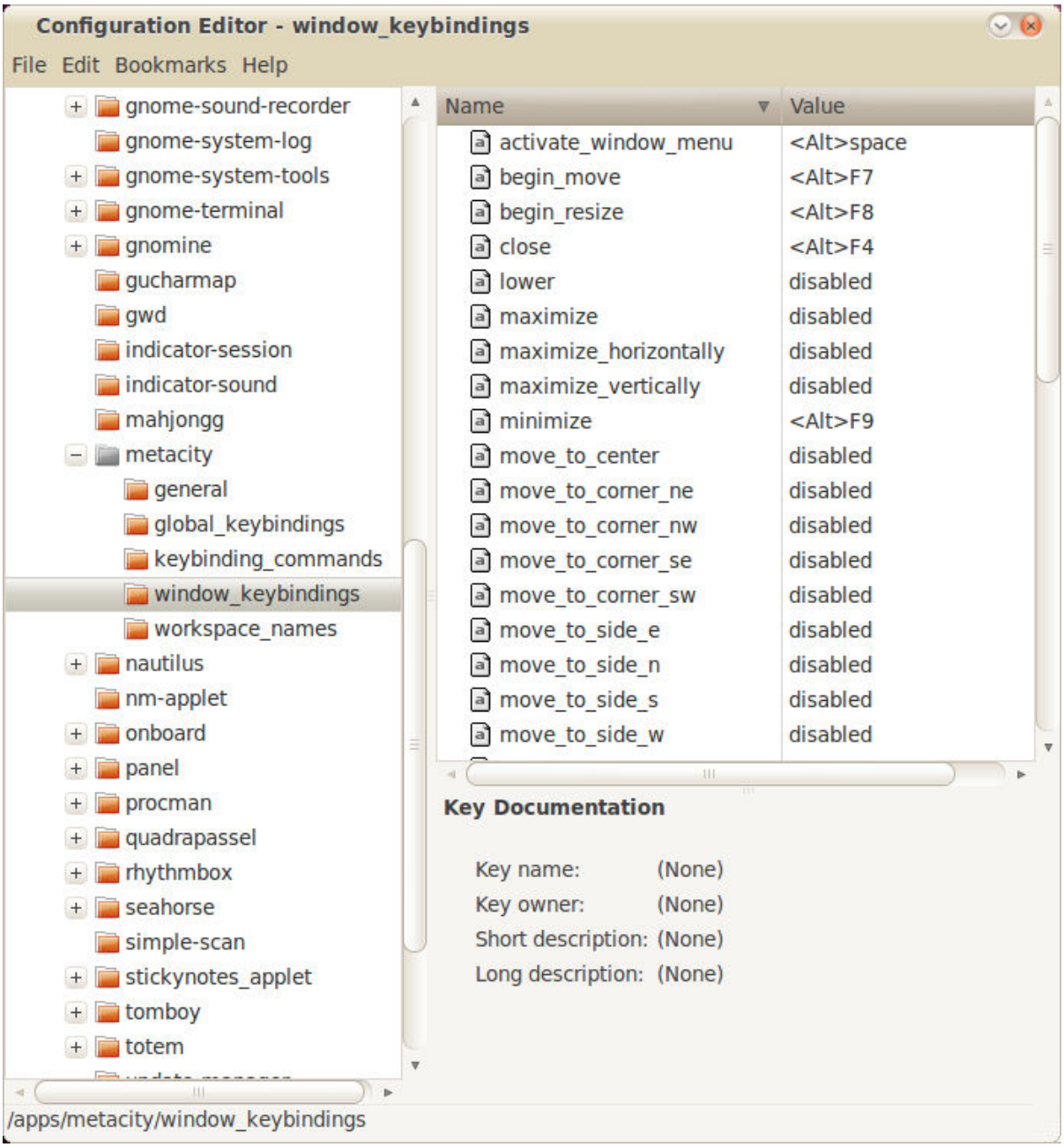
Tretji pogoj je najbolj odvisen od navad samega uporabnika. Vsi pomnimo in obravnavamo informacije na zelo oseben način, enako je z logiko izbire bližnjice za določeno dejanje. Kljub temu lahko razdelimo človeško pomnjenje na vizualno in tekstualno. Zaradi majhnega števila tipk v območju dosegljivosti roke v optimalnem položaju je potrebna njihova kombinacija. Tekstualno pomnjenje bližnjic pomeni, da so izbrane tako, da lahko asociirajo na dejstvo oziroma funkcijo, ki jo izvajajo. Tako bi lahko na primer za zapiranje okna izbrali bližnjico <Alt> + <c> (angl. beseda close, ki pomeni "zapri" in se začne s črko c) ali <Alt> + <x>, saj črka x spominja na podobo "X", ki služi za zapiranje okna. "Vizualno" v načinu izbire bližnjic pomeni "orientacijsko", ker bomo izbirali tipke, katerih položaj je primeren orientaciji nekega določenega dejstva. Na primer, če izberemo bližnjico <Alt> + <q> za premik okna v levi zgornji kot, bomo za premik v desni zgornji kot izbrali bližnjico <Alt> + <w>, ker tipka <w> na tipkovnici leži desno od tipke <q>. Premik v levi spodnji kot bo <Alt> + <a> ipd.

4.3 Namestitev bližnjic

Globalne bližnjice za namizno okolje GNOME najdemo v *Keyboard Shortcuts* v meniju *System – Preferences*. Poleg splošnih sistemskih funkcij, kot so upravljanje zvoka ter poganjanje določenih sistemskih dialogov in aplikacij, imamo tudi možnost namestitve bližnjic za nekatere splošnejše funkcije Metacityja. Te najdemo v oddelku *Window Management*. Dodatne funkcije Metacityja so navadnemu uporabniku skrite zaradi principa KISS, na katerem je zasnovan GNOME [8]. Najdemo jih v Konfiguracijskem urejevalniku (*Configuration Editor*) v oddelku */apps/metacity*.



Slika 7: Globalne bližnjice v namiznem okolju GNOME



Slika 8: Konfiguracijski urejevalnik

Oddelek */apps/metacity/general* vsebuje lastnosti, s pomočjo katerih lahko spremenimo obnašanje Metacityja. V */apps/metacity/global_keybindings* najdemo bližnjice za prebiranje med preglednimi ploščami, odprtimi okni in navideznimi namizji, bližnjice za odpiranje posebnih dialogov namiznega okolja itn. Tukaj je na razpolago tudi 12 bližnjic po meri uporabnika z nastavljivimi ukazi, ki se bodo zagnali ob njihovem klicu. Pripadajoče ukaze lahko nastavimo v oddelku */apps/metacity/keybinding_commands*. Oddelek */apps/metacity/window_keybindings* vsebuje bližnjice za upravljanje oken in */apps/metacity/workspace_names,* imena navideznih namizij.

Zadržali se bomo na *.../global_keybindings* in *.../window_keybindings,* kjer bomo nastavili bližnjice za tista dejanja, ki jih bomo pozneje upravljali tudi s pomočjo gest. Nastavili jih bomo po principu treh pogojev akordične tipkovnice.

Naslednji tabeli prikazujeta funkcije in pripadajoče bližnjice, ki jih bomo izbrali kot primer.

Funkcionalnost	**Bližnjica**
cycle_windows	<Alt>Escape
show_desktop	<Mod4>d
switch_to_workspace_1	<Mod4>1
switch_to_workspace_2	<Mod4>2
switch_to_workspace_3	<Mod4>3
switch_to_workspace_left	<Mod4>q
switch_to_workspace_right	<Mod4>w
switch_windows	<Alt>Tab

Tabela 1: Globalne bližnjice

Funkcionalnost	Bližnjica
close	<Mod4>x
maximize	<Alt><Mod4>m
maximize_horizontally	<Alt><Mod4>c
maximize_vertically	<Alt><Mod4>v
minimize	<Mod4>a
move_to_center	<Alt><Mod4>y
move_to_corner_ne	<Alt><Mod4>q
move_to_corner_nw	<Alt><Mod4>w
move_to_corner_se	<Alt><Mod4>a
move_to_corner_sw	<Alt><Mod4>s
move_to_workspace_1	<Alt><Mod4>1
move_to_workspace_2	<Alt><Mod4>2
move_to_workspace_3	<Alt><Mod4>3
move_to_workspace_left	<Shift><Mod4>q
move_to_workspace_right	<Shift><Mod4>w
raise_or_lower	<Alt><Mod4>d
toggle_above	<Mod4>e
toggle_fulscreen	<Alt><Mod4>f
toggle_maximized	<Alt><Mod4>x
toggle_shaded	<Mod4>s
unmaximize	<Alt><Mod4>n

Tabela 2: Bližnjice za upravljanje oken

Modifikacijska tipka <Mod4> je pravzaprav “Windows tipka” oziroma tipka z Windowsovim logom. V nadaljnjem besedilu jo bomo označevali kot <Win>. Skoraj vsa dejanja se bodo klicala z bližnjicami, ki vsebujejo modifikacijsko tipko <Alt>, <Win> ali njuno kombinacijo, saj sta roki najlažje dostopni. Zaradi lažjega pomnjenja bomo za dejanja, ki kakorkoli spreminjajo dimenzije ali položaj okna, uporabljali kombinacijo <Alt> in <Win>. Za vsa druga dejanja bomo uporabljali le <Win> kot modifikacijsko tipko, razen v primeru preklapljanja med odprtimi okni, kjer bomo obdržali dobro znano bližnjico <Alt> + <Tab>, ki se uporablja v operacijskem sistemu Windows. Podobno tej logiki bomo uporabljali <Alt> + <Escape> za izbiranje med odprtimi okni.

5 GESTE V GNU/LINUXU

Čeprav uporaba gest z miško znotraj operacijskih sistemov ni nikoli zaživela, imamo danes veliko orodij za njihovo interpretacijo. Obstajajo orodja za Mac OS X (FlyGesture, CocoaGestures, xGestures, ...), Windows (Just Gestures, High Sign, Gesture Magic, gMote, ...) in GNU/Linux (Gestikk, wayV, xgestures, xstroke, ...). Nekatere aplikacije imajo vgrajeno podporo za geste, na primer spletni brskalnik Opera ali Firefox z ustreznimi vtičniki. Prav tako so geste prisotne med nekaterimi igrami, ki uporabljajo geste z miško kot način upravljanja (Black & White, Arx Fatalis). Mi se bomo ukvarjali z dvema orodjema: Easystroke za GNU/Linux in StrokeIt za Windows.

5.1 Easystroke

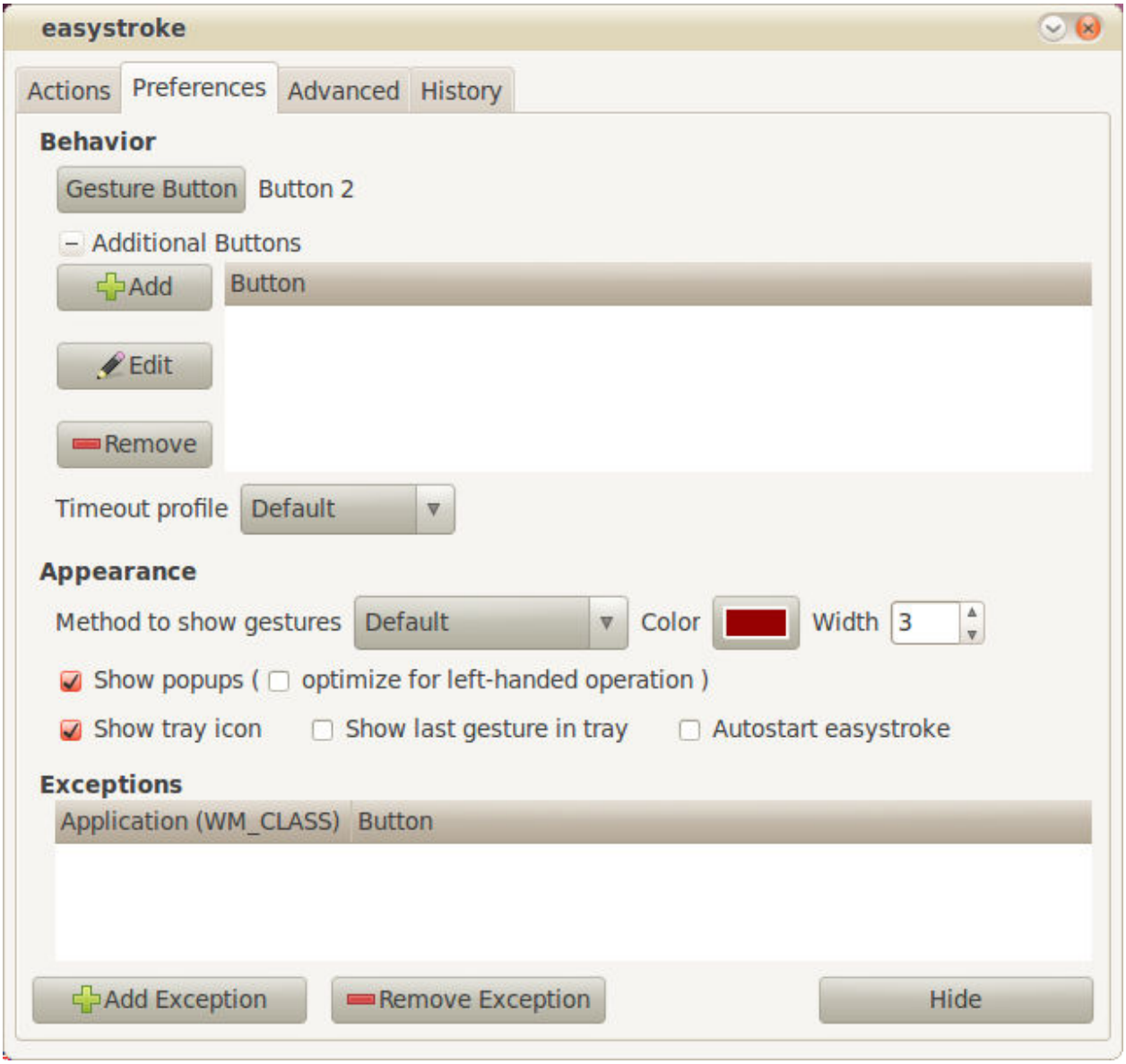
Easystroke je brezplačen odprtokodni program za interpretacijo gest z miško. Izvaja se v ozadju in ima majhen pomnilniški odtis (okrog 6 MB). Iz ozadja čaka na uporabnika, da začne risati geste. Pravzaprav čaka, da uporabnik pritisne določeni sprožilni gumb miške (na začetku je privzet srednji gumb miške) in nato začne "poslušati" vse dogodke, dokler uporabnik ne spusti sprožilnega gumba. Dokler je sprožilni gumb pritisnjen, kazalček med premikanjem pušča za sabo sled in zdi se, kot da na zaslon rišemo simbole. To je za uporabnika zelo koristna povratna informacija, ker v vsakem trenutku vidi, kaj je narisal in kje je kazalček od začetne točke risanja, kar olajša orientacijo. Risanje oziroma vidno sled kazalčka lahko izklopimo, kakor bomo videli kmalu. Konec risanja geste program zazna v trenutku, ko spustimo sprožilni gumb, nato gesto primerja s slikami shranjenih gest v podatkovni bazi. Ko pride do ujemanja, se izvedejo ustrezni ukazi, ki so posebej določeni za vsako shranjeno gesto.

5.2 Zgled programskega vmesnika

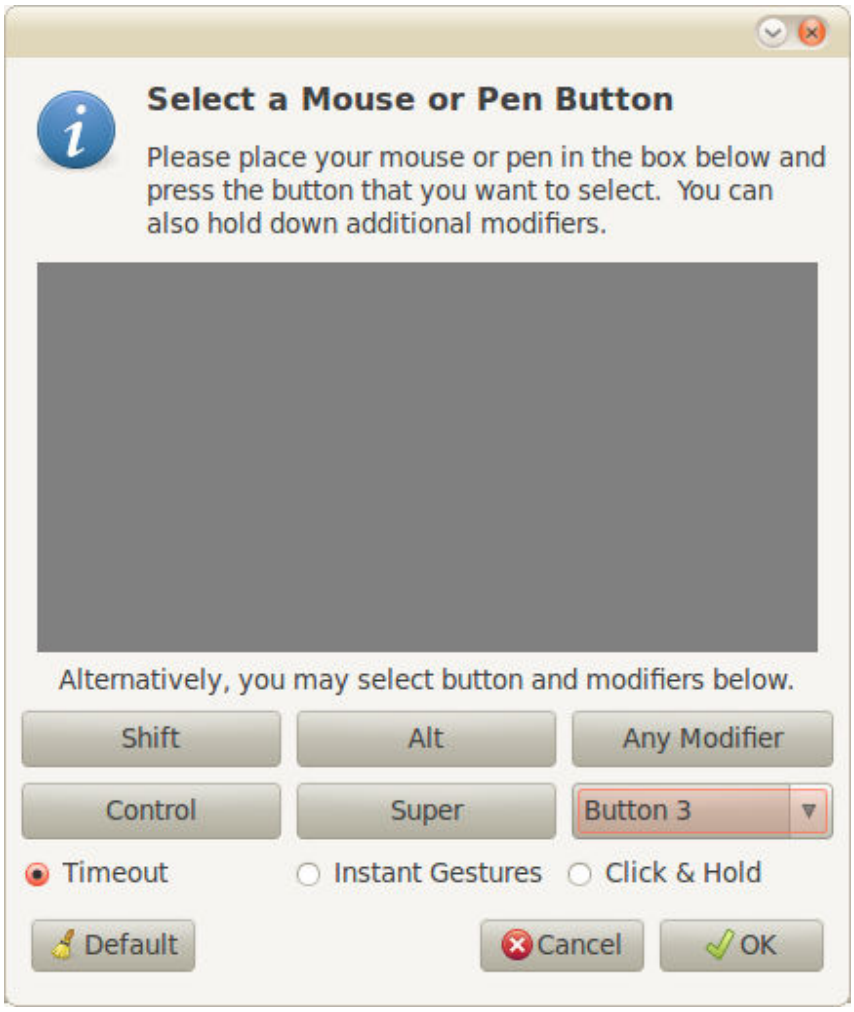
Ko poženemo program, se v obvestilnem območju pojavi ikona (slika 9) in lahko rišemo geste. V tem trenutku se še nič ne zgodi, ker program ne prepozna nobene geste. Z desnim klikom na ikono se odpre meni, s katerim lahko začasno onesposobimo program ali ga popolnoma zapremo.

Levi klik na ikono odpre konfiguracijsko okno. Prvi zavihek (Actions) prikazuje shranjena dejanja s pripadajočimi gestami in ukazi. O tem bo več govora pozneje. V drugem zavihku so izbirne nastavitve (Preferences).

Slika 9: Obvestilno območje



Slika 10: Izbirne nastavitve

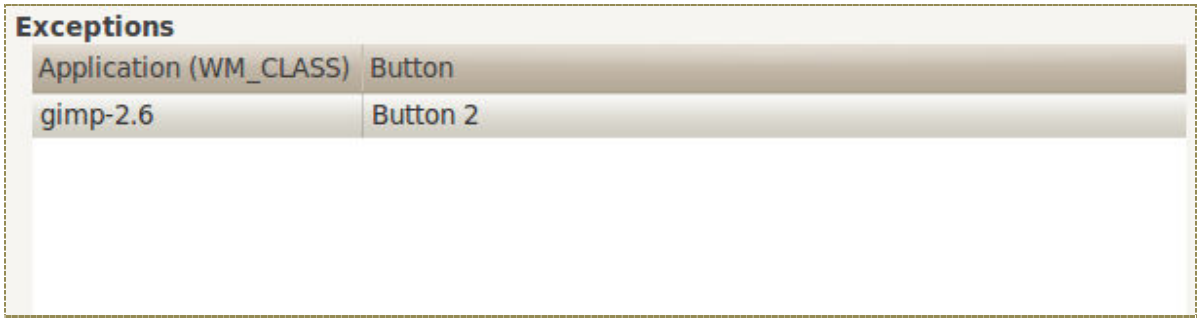


Slika 11: Okno za izbiro gumba

Tukaj lahko spremenimo oziroma nastavimo sprožilni gumb miške. Klik na gumb *Gesture Button* odpre okno za njegovo izbiro (slika 11). Če želimo, lahko izberemo tudi eno ali več modifikacijskih tipk, ki bodo v kombinaciji z gumbom miške sprožile risanje gest. Easystroke je poseben v tem, da ima lahko več sprožilnih gumbov. Delimo jih na tri vrste: navadni (*Timeout*), instantni (*Instant Gestures*) in zadržani (*Click & Hold*). Navadni sprožilni gumb edini omogoča risanje gest in je lahko časovno omejen. Instantni gumb služi za sprožanje izključno naprednih

gest, o katerih bo več govora pozneje. Zadržani gumb lahko samostojno sproži dejanje, če ga nekaj časa držimo pritisnjenega. Lahko ga uporabljamo tudi kot instantnega. V oddelku za dodatne gumbe *Additional Buttons* lahko dodamo dodatne sprožilne gumbe. Spustni meni *Timeout profile* služi za izbiro nastavitve, ki določa, kako hitro bo navadni sprožilni gumb prenehal biti aktiven, če miška miruje. Če je časovna omejenost izklopljena (*Timeout off*), je gumb aktiven, dokler ga ne pustimo. Sledi, ki jo pušča kazalček pri risanju gest, lahko spremenimo barvo (*Color*) in širino (*Width*). Način, na katerega sistem prikazuje sled, izberemo v spustnem meniju *Method to show gestures*; na tem mestu ga lahko popolnoma izklopimo z izbiro *None*. Če vklopimo nastavitev *Show popups*, bo Easystroke, vsakič, ko uporabimo kakšno gesto, poleg kazalčka prikazal pojavno sliko in ime geste, ki jo je prepoznal. To je odlično in hitro povratno obvestilo, ki nam pove, kaj smo pravkar naredili (slika 12).

Slika 12: Obvestilo o prepoznani gesti

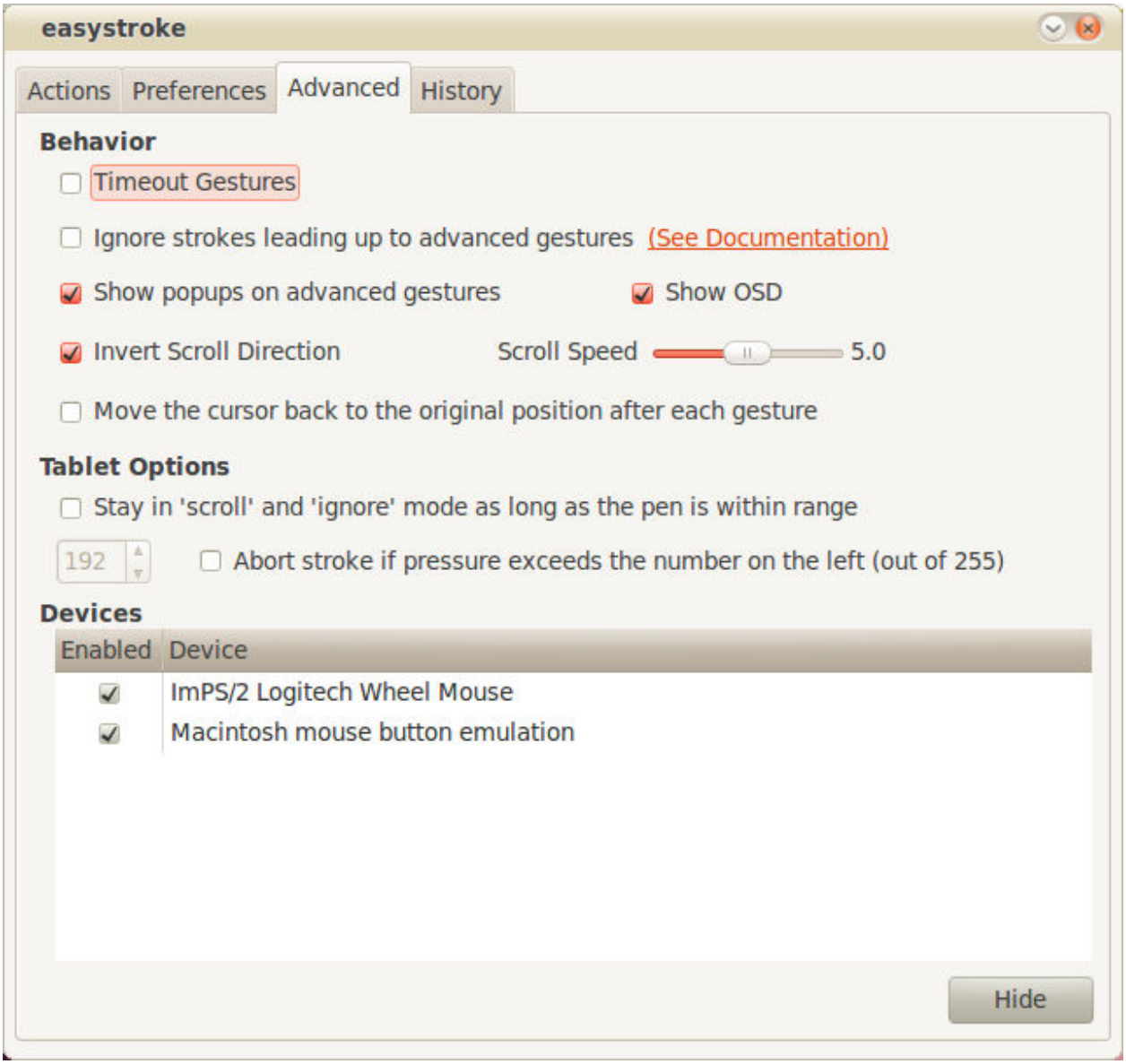


Slika 13: Seznam izjem

Show last gesture in tray prikazuje sliko zadnje narisane geste v obvestilnem območju namesto ikone programa. Aplikacije, znotraj katerih bo Easystroke onesposobljen, najdemo v seznamu izjem (*Exceptions)*. Določeno aplikacijo dodamo z gumbom *Add Exception* in potem kliknemo na njeno okno na namizju.

Tudi v primeru, da ni aplikacij, znotraj katerih nočemo, da Easystroke deluje, lahko obstajajo takšne, ki nujno potrebujejo Easystrokov rezerviran sprožilni gumb. Primer takšne situacije bi bil, če imamo kot sprožilni gumb nastavljen desni gumb miške in radi uporabljamo GIMP. V GIMP-u se desni gumb uporablja znotraj nekaterih orodij za obdelavo slik in zato nočemo, da ga Easystroke rezervira samo zase. Težavo odstranimo tako, da GIMP dodamo na seznam *Exceptions* in mu priredimo nadomestni sprožilni gumb, recimo srednji gumb miške (slika 13).

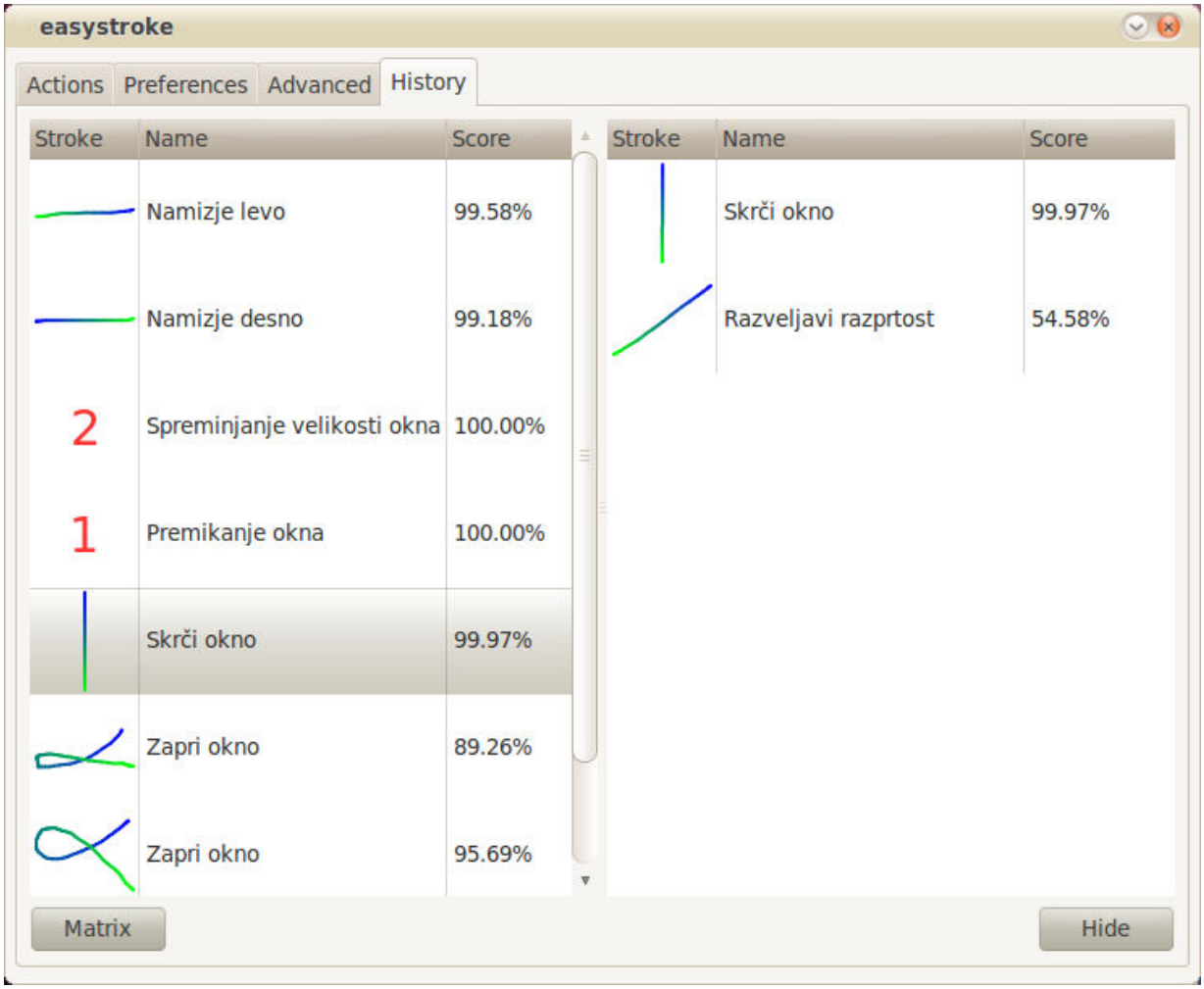
V tretjem zavihku (*Advanced*) najdemo napredne nastavitve (slika 14). Nastavitev *Ignore strokes leading up to advanced gestures* pomeni ignoriranje potez znotraj naprednih gest. Poteze v naprednih gestah Easystroke ne bo obravnaval, če je nastavitev omogočena, kar pravzaprav pomeni, da bo število uporabnih naprednih gest enako številu gumbov miške razen sprožilnega. Več o naprednih gestah bomo govorili v posebnem poglavju. Če je omogočena nastavitev Show *OSD*, bodo v obliki "zaslonskega obvestila" (angl. On-Screen Display, OSD) prikazani nazivi modifikacijskih gumbov, ki so aktivirani, dokler držimo sprožilnega ob trajanju geste pritisnjenega. *Invert Scroll Direction* zamenja smer, *Scroll Speed* pa spremeni hitrost v gesti za pomikanje (*Scroll*). *Tablet Options* so nastavitve, ki se nanašajo izključno na tablične računalnike in grafične tablice. Na koncu je seznam enot (*Devices*), ki se uporabljajo za premikanje kazalčka po zaslonu. Če odstranimo kljukico z ene izmed njih, je Easystroke ne bo obravnaval kot vhodno napravo za risanje gest.



Slika 14: Napredne nastavitve

Četrti zavihek (*History*) shranjuje zadnje narisane geste in služi kot dnevnik uporabe gest (slika 15). Ko narišemo novo gesto v sistemu, se ta pojavi na seznamu na levi strani. Poleg slike geste, ki smo jo narisali, sta v primeru ujemanja ime shranjene geste in odstotek ujemanja. Na seznamu na desni strani so prikazane vse shranjene geste, ki se z našo ujemajo za več kot 50 %. Dnevnik je lahko zelo koristen dodatek, saj nam omogoča, da lažje spremljamo uporabo gest v sistemu. Če

se zgodi, da se program zmoti in zamenja eno gesto za drugo, lahko odpremo dnevnik in ugotovimo, zakaj je prišlo do napake ter katere geste so si preveč podobne. Nato se lažje odločimo, katere shranjene geste je treba spremeniti in na kakšen način.



Slika 15: Dnevnik uporabe gest

5.3 Konfiguriranje gest

Geste s pripadajočimi ukazi sestavljajo dejanja. Ko uporabimo gesto, nastane dejanje, ki poda sistemu ustrezen ukaz. Obstajata dve osnovni skupini gest: enostavne in napredne. Obe vrsti bomo v nadaljnjem besedilu obrazložili s konkretnimi primeri, za katere bomo uporabljali desni gumb miške kot glavni sprožilni gumb.

5.3.1 Enostavne geste

Določanje samih gest bomo prikazali s primerom, v katerem bomo geste izbirali tako, da čim bolj spoštujemo naslednja načela:

1. geste naj bodo čim hitrejše in enostavne za risanje,
2. geste naj bodo čim bolj intuitivne,
3. geste naj se čim bolj razlikujejo med seboj.

Pri prvi točki je priporočljivo, da imajo pogostejša dejanja enostavnejše geste kot manj pogosta dejanja.

Intuitivnost v drugi točki bomo dosegli podobno kot pri konfiguriranju bližnjic v Metacityju. Uporabili bomo aspekta simbolične in orientacijske interpretacije. Dejanjem, ki premikajo oziroma širijo okno v določeno smer, bodo pridružene geste, usmerjene v enako smer. Na primer premik okna v spodnji desni kot bomo dosegli s poševno črto iz središča proti spodnjemu desnemu kotu namizja. Podobno temu se bomo premaknili na desno sosedno navidezno namizje z ravno črto od leve proti desni. Simbolični aspekt uporabimo tako, da rišemo simbole, ki so povezani z namenom, ki ga želimo doseči. Na primer kopiranje (*copy*) dosežemo z gesto v obliki črke “c”, lepljenje (*paste*) pa z gesto v obliki črke “p”. Ker smo že uporabili gesto v obliki črke “c” za zapiranje okna, bomo uporabili gesto oblike “x”, ker ta spominja na podobo za zapiranje okna.

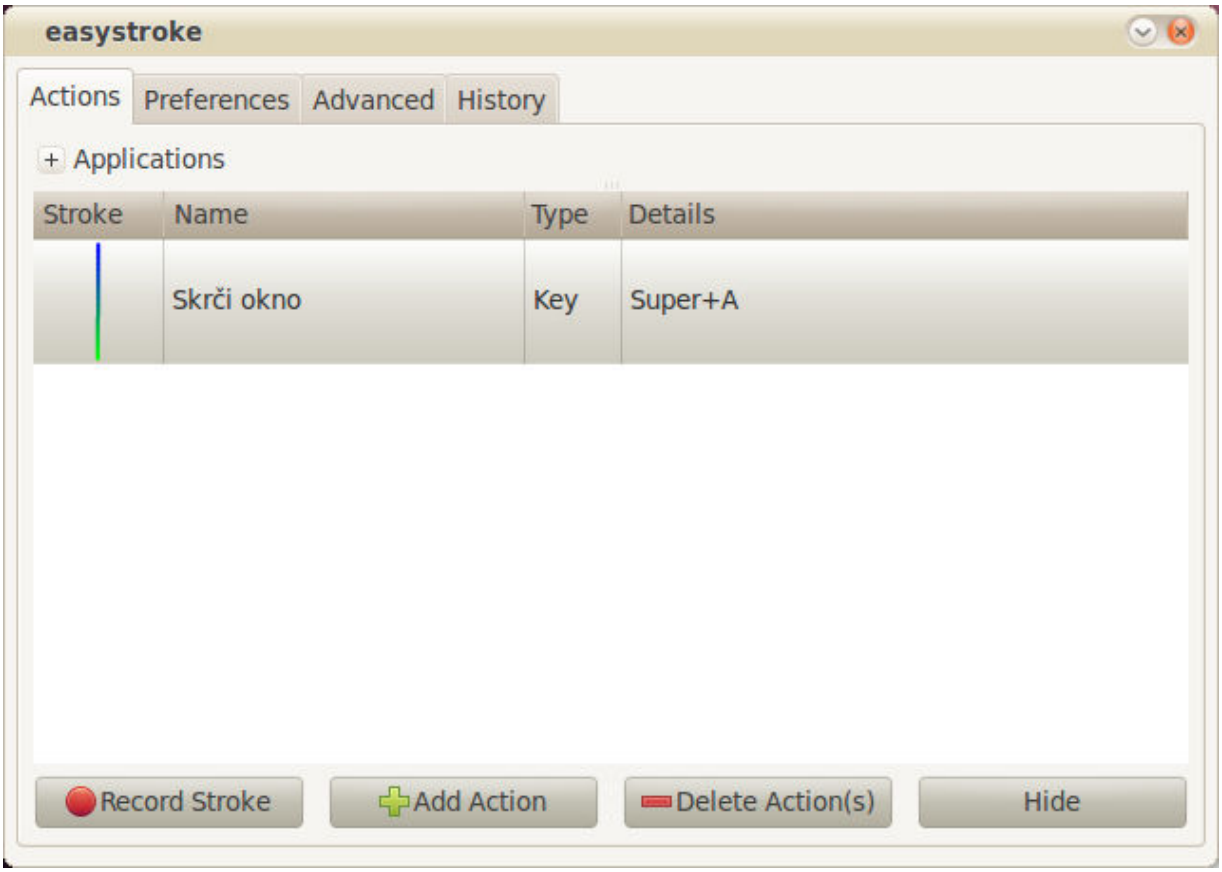
Tretja točka preprečuje, da bi se program zmotil in v prepoznavanju zamenjal dve gesti ter izvedel napačen ukaz.

Za dodajanje novega dejanja na seznam kliknemo gumb *Add Action*. Novo dejanje ima ime "Gesture 1". Zaradi lažjega orientiranja bomo ustrezno spremenili imena. Prvo dejanje se bo imenovalo "Skrči okno". V naslednjem koraku bomo označili tip ukaza, ki ga sproži sproži v sistemu. Easystroke podpira naslednje tipe ukazov:

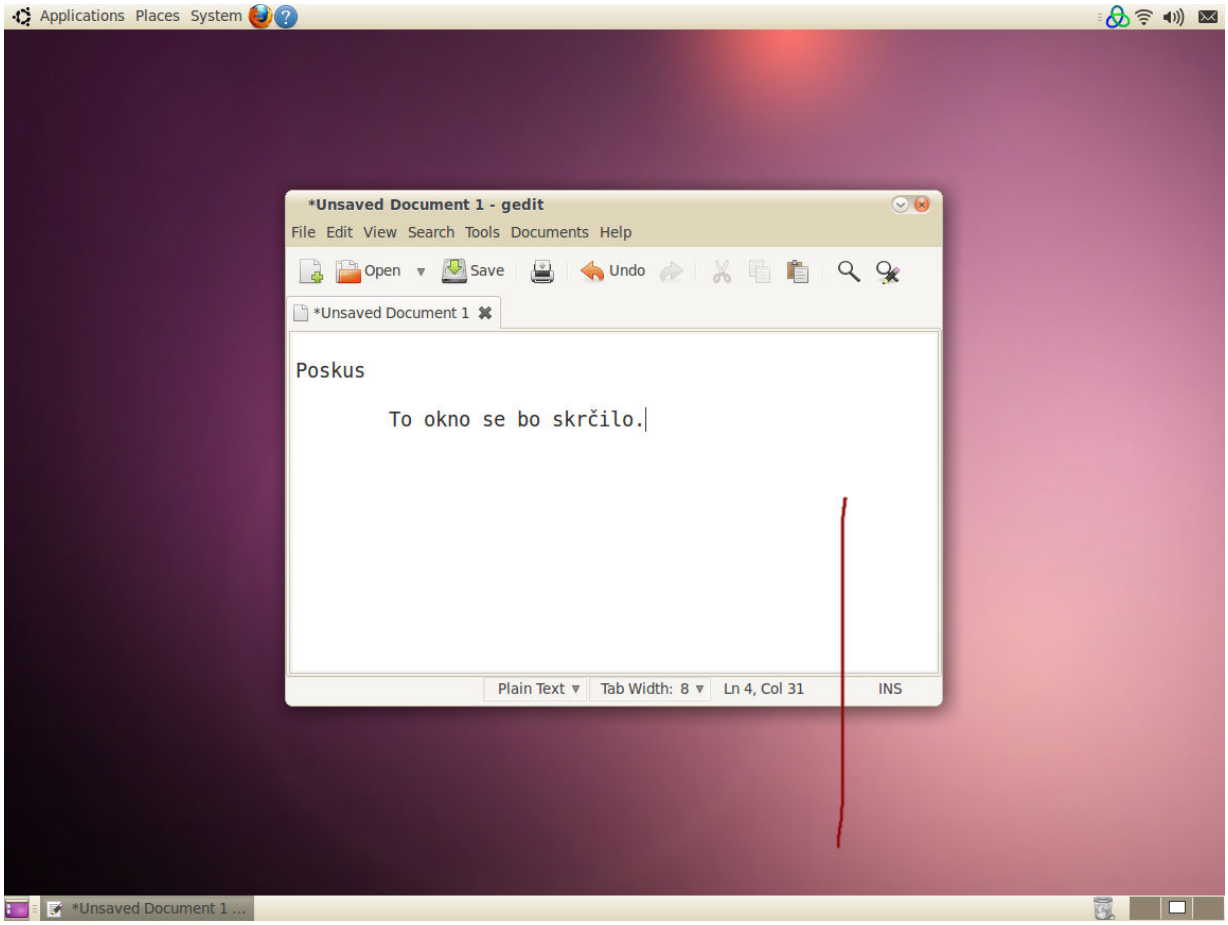
- ukazni stavek (*Command*),
- kombinacija tipk (*Key*),
- vpis besedila (*Text*),
- pomikanje vsebine (*Scroll*),
- ignoriranje geste (*Ignore*),
- klik gumba miške (*Button*),
- razno (*Misc*).

Za novo dejanje izberemo ukaz Key. Uporabljali bomo bližnjice, ki smo jih prej namestili v Metacity konfiguraciji. Za skrčitev okna bomo torej uporabili bližnjico <Win> + <a>. Bližnjico vnesemo v dejanje v polju *Details*. Zdaj je treba samo še določiti gesto. Kliknemo na gumb Record Stroke in narišemo navpično črto od zgoraj navzdol. Easystroke si takoj zapomni spremembo in sliko geste, ki smo jo narisali, prikaže levo od imena. Ta korak ponavljamo, dokler nismo zadovoljni z izidom. Geste rišemo normalno, tako kot bi jih sicer narisali v kateremkoli trenutku. Pomembno je, da risba čim bolj ustreza naši naravni izvedbi in da ne strmimo k čim lepši ali poravnani črti. Ko končamo, je naša prva gesta nameščena. Za poskus odpremo katerokoli okno in prek njega narišemo navpično črto navzdol. Okno bi se moralo skrčiti in ostati v opravilni vrstici. Pri uporabi gest nista pomembna ne velikost risbe ne hitrost risanja.

Pomembno je le, da leži točka, v kateri začnemo risati, kjerkoli znotraj ciljanega okna. Risba lahko presega robove okna, kot to prikazuje slika 17.

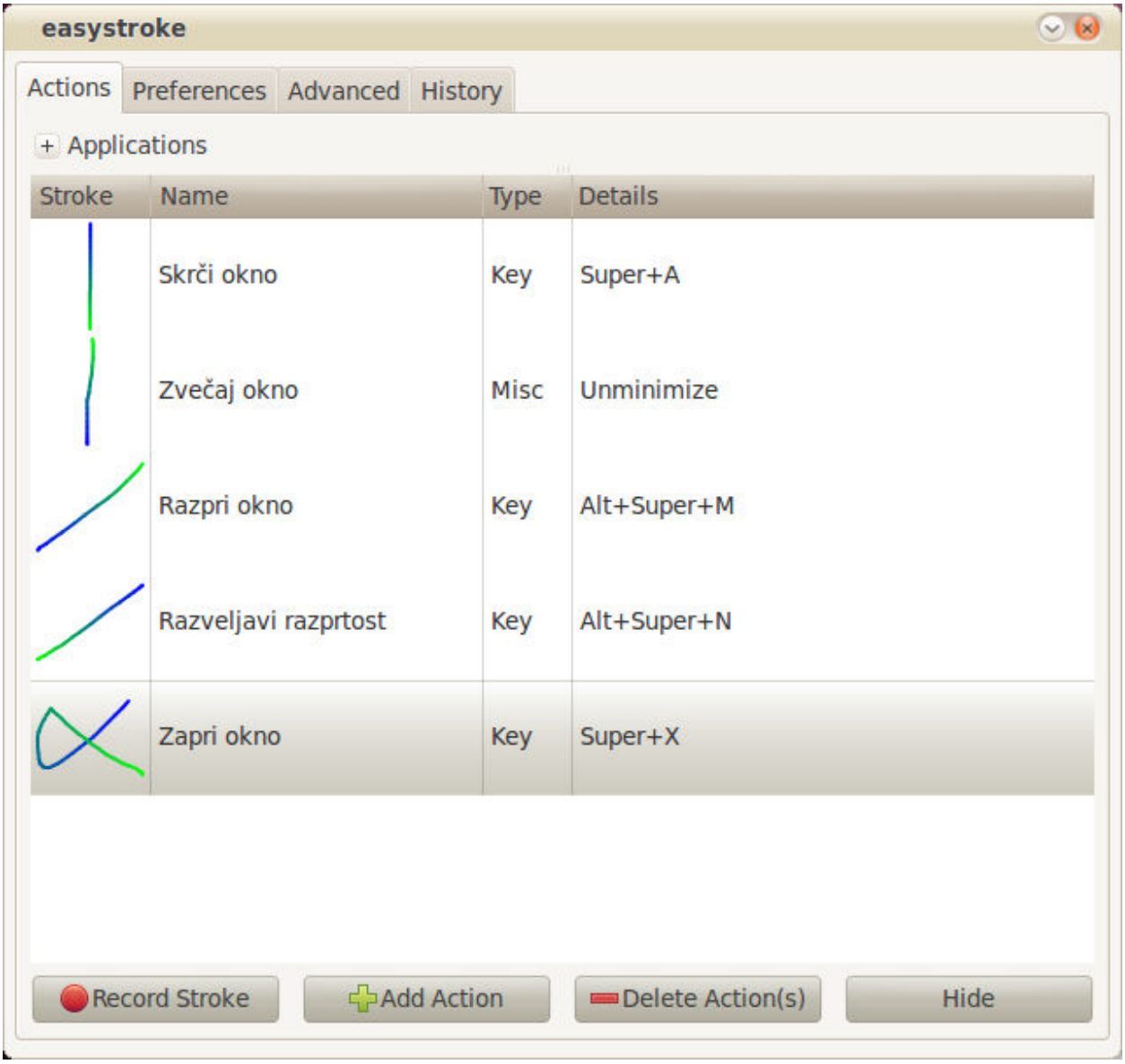


Slika 16: Nameščanje nove geste



Slika 17: Dovoljen način risanja gest

Easystroke ima vgrajeno možnost povečanja oken v obrnjenem vrstnem redu od njihovega skrčenja (LIFO). Naša naslednja gesta bo izkoristila prav to možnost. Izberemo *Misc* kot tip ukaza, v polju *Details* pa *Unminimize*. Gesto narišemo obrnjeno kot za skrčitev okna, torej navpično črto navzgor. Gesta za razpiranje okna bo poševna črta proti zgornjemu desnemu kotu, gesta za vračanje v predhodno stanje oziroma razveljavitev razprtosti (*Unmaximize*) pa bo spet obrnjena gesta, torej poševna črta iz središča proti spodnjemu levemu kotu. Easystroke ne podpira samo ravnih črt. Narišemo lahko tudi zlomljene ali ukrivljene črte. Gesto za zapiranje okna bomo narisali v obliki črke "X" oziroma bolj podobno grški črki alfa (α), ker lahko rišemo samo v eni potezi. Z enako logiko namestimo tudi preostale geste za dejanja z okni (slika 18).

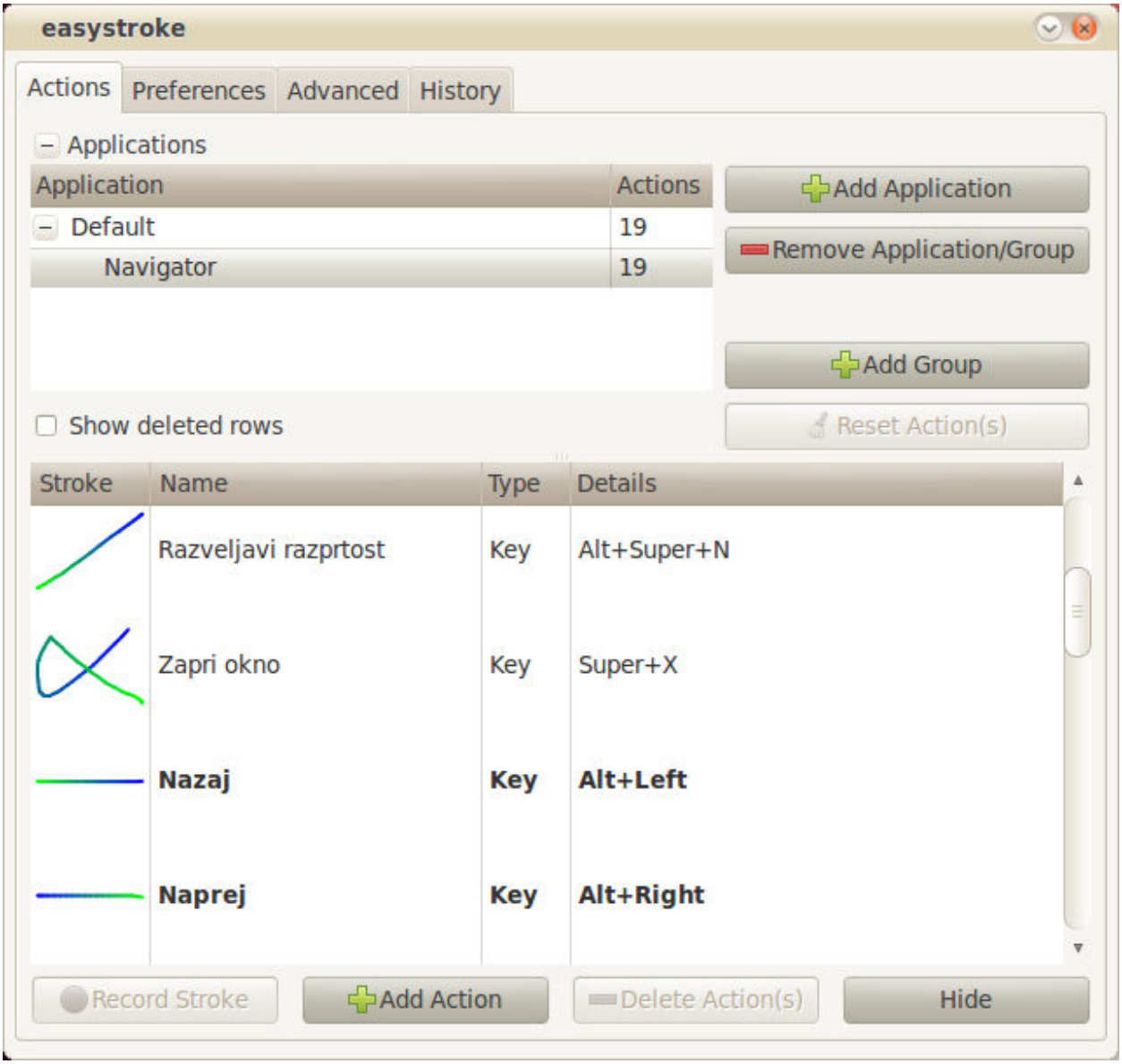


Slika 18: Še nekaj gest za dejanja z okni

Do težav pride, ko hočemo uporabiti isto logiko oblikovanja gest za dejanja iste orientacije. Na primer, če hočemo imeti ravno črto z desne proti levi za prehod na levo sosednje navidezno namizje in podobno temu tudi gesto za nazaj (*Back*), recimo v spletnem brskalniku. Res je, lahko bi si izmislili nekoliko drugačno oziroma bolj zapleteno gesto, vendar to ne bi bilo v skladu z idejo iz prvega in drugega načela za določanje gest. Treba je preprečiti zmedo med zelo

podobnimi gestami. Eden od načinov je, da uporabljamo isto gesto, ki ji predhodno spremenimo dejanje v eni oziroma več ciljnih aplikacijah. To lahko dosežemo v oddelku *Applications*, ki ga najdemo nad seznamom gest. Na začetku imamo eno prazno skupino *Default*. Novo skupino dodamo z gumbom *Add* Group. Za dodajanje aplikacije kliknemo gumb *Add Application* in potem kliknemo okno izbrane aplikacije na namizju. Dodana aplikacija podeduje vsa dejanja skupine, v kateri je. Posameznim aplikacijam in tudi celim skupinam lahko spremenimo ukaz, ki ga bo povzročila posamezna gesta. Prav tako je mogoče posameznemu dejanju spremeniti oziroma izbrati novo gesto, ki bo delovala samo znotraj označene aplikacije oziroma skupine aplikacij. Spremembe so v tem primeru na seznamu gest skupine/aplikacije označene s krepko pisavo. Če nismo zadovoljni s spremembami, jih lahko razveljavimo z gumbom *Reset Action(s)*.

Za naš primer naredimo dve novi dejanji, enega za prehod na levo sosednje navidezno namizje in drugega za prehod na desno. Za tip njunega ukaza izberemo *Key*, za bližnjici pa <Win> + <q> za levo in <Win> + <w> za desno namizje, tako kot smo že nastavili v Metacityju. Ko smo to storili, kliknemo na gumb za dodajanje aplikacije, kjer dodamo naš spletni brskalnik. Nato na njegovem seznamu spremenimo ukaza dejanj za prehode v sosednji namizji tako, da jima nastavimo novi bližnjici, in sicer <Alt> + <Left> in <Alt> + <Right>. To sta standardni bližnjici za sprehajanje nazaj/naprej po zgodovini. Ravna črta v levo nas bo tako prestavila na levo namizje, z izjemo klica v oknu spletnega brskalnika, kjer bo le vrnila brskalnik na zadnjo obiskano stran (slika19). To seveda ni najelegantnejša rešitev, ker bomo morali vsakič pri premiku na sosednje navidezno namizje paziti, da geste ne rišemo v spletnem brskalniku. V Easystroku obstaja v ta namen veliko elegantnejša rešitev, ki se imenuje napredne geste (angl. Advanced Gestures).



Slika 19: Spremembe dejanj gest v določenih aplikacijah

5.3.2 Napredne geste

Napredne geste v Easystroku so tiste, pri katerih uporabimo še enega od drugih gumbov, medtem ko je pritisnjen sprožilni gumb. Če ni nastavljena možnost ignoriranja potez znotraj naprednih gest, lahko z njihovo uporabo povečamo skupno število uporabnih gest brez nepotrebnega pomnjenja dodatnih gest. Glavni pomen naprednih gest je ta, da z uporabo samo enega dodatnega gumba miške spremenimo eno gesto v popolnoma drugačno, čeprav sta obe zasnovani na istih potezah. To praktično pomeni, da si ni treba izmisliti novih, popolnoma drugačnih gest z zapletenejšimi oblikami potez, temveč lahko vzamemo ustrezno že obstoječo gesto in jo predelamo v novo, ki se razlikuje zgolj v enem dodatnem kliku. Na tak način obdržimo smernice iz prvih dveh načel ter intuitivnost in enostavnost pa tudi tretji cilj, to je dovolj veliko raznolikost, da ne pride do napačnega prepoznavanja geste. Dodaten klik gumba lahko razumemo kot modifikacijsko tipko v bližnjicah na tipkovnici. Kakor z uporabo recimo tipke <Alt> spremenimo katerokoli bližnjico v popolnoma novo, tako jo s klikom na gumb ob koncu geste spremenimo v popolnoma drugačno.

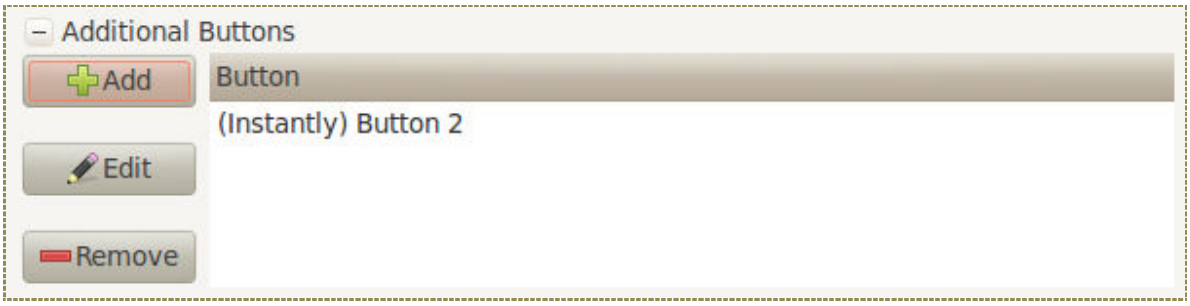
Obstajajo trije osnovni tipi naprednih gest:

1	Gesta, pri kateri sta oba gumba pritisnjena (sprožilni in levi kot dodatni) brez premikanja miške.
1	Gesta s klikom dodatnega gumba na koncu risanja poteze.
×	Simbol označuje en sam klik sprožilnega gumba.

Tabela 3: Trije tipi gest

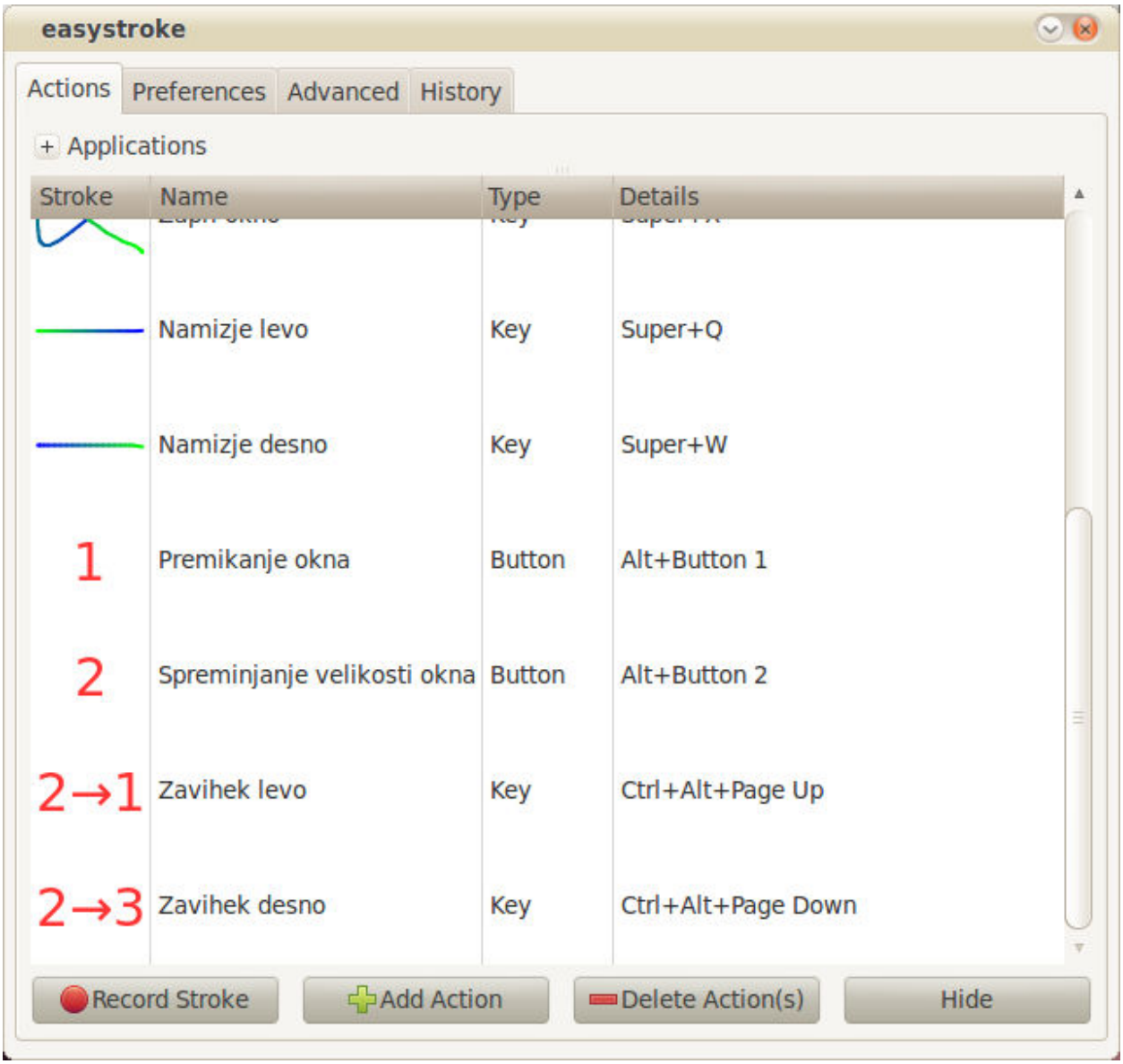
Navedli bomo nekaj primerov za prvi tip. Gesta za dejanje “premikanje okna” bo kombinacija sprožilnega in dodatnega levega gumba. Dejanje bo kot ukaz izvršil klik levega gumba miške v kombinaciji z modifikacijsko tipko <Alt>. Gesta za “spreminjanje velikosti okna” bo kombinacija sprožilnega in srednjega gumba. Ukaz, ki ga bo sprožila, bo klik srednjega gumba

miške v kombinaciji z <Alt>. Zdaj lahko premikamo okna in jim spreminjamo velikost brez uporabe tipke <Alt> na tipkovnici. Easystroke "drži" <Alt> namesto nas, pri čemer mi poskrbimo samo za premik okna oziroma za to, koliko ga bomo povečali. Če je nastavitev "*Show OSD*" omogočena, nas bo Easystroke v zgornjem desnem kotu zaslona obvestil o tem, da je tipka <Alt> aktivirana (slika 22). Naslednji dve gesti bosta nekoliko drugačni. Sprožil ju bo novi sprožilni gumb, posebej konfiguriran zanju. Naš novi sprožilni gumb dodamo v oddelku za dodatne gumbe v izbirnih nastavitvah. Za ta primer bo to srednji gumb miške, ki bo instantnega tipa.

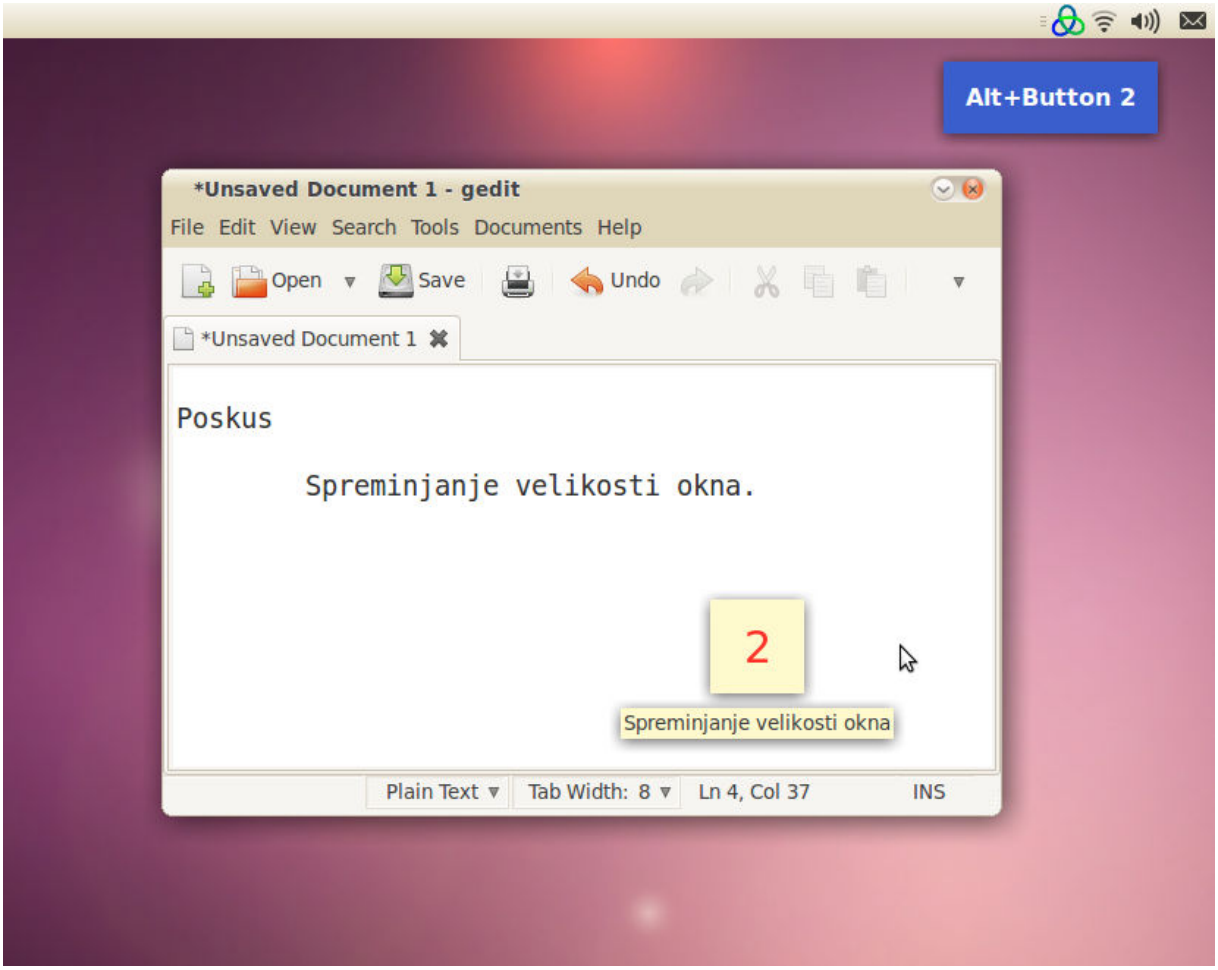


Slika 20: Dodatni sprožilni gumb

Eno dejanje bo "zavihek levo". Sprožila ga bo kombinacija srednjega (zdaj sprožilnega) in levega gumba. Kot dejanje namestimo <Ctrl> + <Page Up>, kar je sistemska bližnjica za levi sosednji zavihek. Drugo dejanje bo "zavihek desno". Tega bo sprožila kombinacija srednjega in desnega gumba, bližnjica pa bo <Ctrl> + <Page Down>. Lahko opazimo razliko med simboloma za na primer "premikanje okna" in "zavihek levo" (slika 21). V prvem primeru je prikazan le drugi gumb, ki pride v kombinaciji z glavnim sprožilnim gumbom, v drugem primeru pa je s puščico označen prehod s srednjega na levi gumb, kar ponazarja, da uporabljamo alternativni, instantni sprožilni gumb, ki ga je treba pritisniti prvega.



Slika 21: Napredne geste 1. tipa



Slika 22: Spreminjanje velikosti

Naslednji tip bomo prikazali z rešitvijo problema podobnih gest iz predhodnega poglavja. Razveljavili bomo spremenjeno delovanje gest z ravno črto v levo/desno znotraj okna spletnega brskalnika. Namesto tega bomo naredili dve novi gesti, popolnoma enaki gestama za prehod na sosednji navidezni namizji levo/desno. Razlika bo le v dodatnem kliku na koncu. Novo gesto za *nazaj* bomo pripravili kot ravno potezo v levo in klik levega gumba miške na koncu poteze (torej preden spustimo sprožilni gumb). Ta gesta bo napredna gesta, ki ji bomo za ukaz nastavili klic bližnjice <Alt> + <Left> za vrnitev na prejšnji položaj v zgodovini. Podobno bomo storili za gesto *naprej*. Ko bomo to končali, bomo imeli dve novi gesti, ki si ju bo zelo lahko zapomniti. Gesta za vrnitev nazaj v zgodovino brskanja bo zasnovana na istih potezah kot gesta za prehod na levo navidezno namizje (slika 23). S tem smo dosegli, da lahko dve različni dejanji naredimo z

dvakrat manjšo kognitivno obremenitvijo, saj imata obe risbi enako obliko, kar omogoča hitrejše pomnjenje. Lahko bi rekli, da smo ubili dve muhi na en mah.

	Namizje levo	Key	Super+Q
	Namizje desno	Key	Super+W
	Nazaj	Key	Alt+Left
	Naprej	Key	Alt+Right

Slika 23: Različne geste z enakimi potezami

	Okno -> zgoraj levo	Key	Alt+Super+Q
	Okno -> zgoraj desno	Key	Alt+Super+W
	Okno -> spodaj levo	Key	Alt+Super+A
	Okno -> spodaj desno	Key	Alt+Super+S
	Okno -> središče	Key	Alt+Super+Y

Slika 24: Še nekaj primerov naprednih gest 2. tipa

Drugi tip naprednih gest lahko uporabimo tudi za premike okna v oglišča in središče namizja (slika 24). Kako uporabiti klik enega samega (sprožilnega) gumba kot gesto in ga pri tem ne omejiti zgolj na Easystroke in mu tako nehote onemogočiti privzeto delovanje v sistemu? Tu nam priskoči na pomoč tretji tip naprednih gest, ki bo dosežen z dolgim klikom gumba. Naredili bomo dejanje za pomikanje vsebine (*Scroll*), ki ga bo sprožil dolg pritisk na srednji gumb miške[1]. Ta gumb imamo že prijavljen kot dodatni sprožilni gumb, ki mu je treba spremeniti vrsto. V oddelku za dodatne gumbe izberemo prej nastavljeni srednji gumb miške in kliknemo gumb *Edit*. Namesto radijskega gumba *Instant Gestures* izberemo *Click & Hold* ter kliknemo *Ok*. Srednji gumb bo obdržal funkcionalnost instantnega gumba, ki ga potrebujemo za prej nastavljene geste, dodatno pa ga bo mogoče uporabiti za geste, ki zahtevajo dolgi klik. Ko smo kliknili gumb *Record Stroke*, pritisnemo srednji gumb miške, ga držimo eno sekundo in spustimo. Na mestu slike geste bo simbol, ki označuje klik srednjega gumba miške (kot dodatnega sprožilnega), ki prehaja v stanje držanja (slika 25). Treba je izbrati še *Scroll* kot tip ukaza in dejanje je nameščeno.

Slika 25: Napredna gesta 3. tipa

Posebnost naprednih gest leži predvsem v tem, da z njimi lahko storimo dejanja, ki niso trenutna (na primer poganjanje nekega ukaza), ampak zahtevajo daljše izvajanje z interakcijo uporabnika (na primer premikanje okna ali pomikanje vsebine). Če se moramo zadržati v nekem interaktivnemu načinu dela, da bi opravili nalogo – takšen je na primer *Scroll* – moramo najprej spremeniti stanje nekega gumba, da bi sploh vklopili ta način dela. V primeru, ko je sprožilni gumb stikalo, ki povzroči interpretiranje geste (in s tem dejanje), je rezultat le trenutno, avtomatično in neodvisno dejanje, saj Easystroke v tem trenutku ne sledi več vhodnim spremembam uporabnika. Ko pritisnemo gumb miške (gumb pride v pritisnjeno stanje), prejme sistem dogodek *button_press_event* [10]. Easystroke čaka, dokler takšnega dogodka ne povzroči sprožilni gumb, saj šele takrat začne "snemati" geste. Ko spustimo sprožilni gumb (gumb pride v

[1] Tip dejanja *Scroll* ni neposredno premikanje vsebine s pomočjo sistemskih klicev, ampak simulira pomikanje miškinega koleščka, dokler premikamo miško navzgor oziroma navdol. Če potrebujemo, lahko v dejanje dodamo kakšno modifikacijsko tipko, ki bo aktivna skupaj z miškinim koleščkom.

dvignjeno stanje), Easystroke prestreže dogodek *button_release_event* [10], prepozna konec geste in začne interpretirati gesto ter izvajati ustrezen ukaz. Če želimo, da Easystroke po tem, ko interpretira gesto, nadaljuje sledenje vhodnim uporabnikovim spremembam, je treba vnesti gumb, ki namesto sprožilnega gumba povzroči interpretiranje geste. Takšen primer smo imeli pri gesti za premikanje okna: vmesni gumb je bil levi. Sprožilni gumb takrat obdrži le vlogo vzdrževanja dejanja oziroma načina dela, v katerem smo. Natančnejši opis bi bil:

1. Easystroke prejme *button_press_event* sprožilnega gumba in začne "snemati" gesto.
2. Easystroke prejme *button_press_event* vmesnega gumba, zaključi vnašanje geste in začne interpretirati gesto.
3. Easystroke preide v interaktiven način delovanja in sledi uporabnikovim spremembam.
4. Easystroke konča dejanje ko prejme *button_release_event vmesnega ali sprožilnega gumba.*

Če vmesni gumb po koncu geste še naprej držimo pritisnjen, lahko v interaktivnem načinu premikamo okna, pomikamo vsebino, spreminjamo velikost okna ipd. Pri naprednih gestah prvega in drugega tipa je vmesni gumb tisti, ki ga pritisnemo po sprožilnem. Pri gestah tretjega tipa je sprožilni in vmesni gumb en sam.

Dejanja, ki uporabljajo daljšo interakcijo z uporabnikom, je možno ustvariti tudi z uporabo navadnih gest. Ignoriranje geste je tip lažnega dejanja, v katerem Easystroke ne stori nič, ko "prebere" gesto, ampak jo naprej poda v okno aplikacije. To je uporabno v primeru, da ima omenjena aplikacija lasten sistem za interpretacijo gest, ki uporablja isti sprožilni gumb, in ne želimo, da ji Easystroke ukrade določene geste. *Ignore* ima tudi drugo možnost. Lahko mu kot dejanje nastavimo aktivacijo ene ali več modifikacijskih tipk. Ko uporabimo gesto, se izbrane modifikacijske tipke aktivirajo in ostanejo aktivne, dokler ne kliknemo kateregakoli gumba miške. Ko takrat pritisnemo katerikoli gumb (razen sprožilnega), Easystroke začne prej opisano interaktivno dejanje in ostane v njem, dokler ne spustimo gumb. Postopek se izvaja tako:

1. Easystroke prejme *button_press_event* sprožilnega gumba in začne "snemati" gesto.
2. Easystroke prejme *button_release_event* sprožilnega gumba, zaključi vnašanje geste, interpretira gesto in preide v stanje čakanja na naslednji gumb.
3. Easystroke prejme *button_press_event* drugega gumba, preide v interaktiven način delovanja in sledi spremembam s strani uporabnika.
4. Easystroke konča izvajanje dejanja, ko prejme *button_release_event* drugega gumba.

Bistvo tega načina je, da lahko sprožimo in potem naprej uporabljamo dejanje v interaktivnem načinu z držanjem le enega gumba (pri naprednih gestah smo morali držati vmesni in sprožilni gumb skupaj). Ko prvič narišemo gesto, program čaka na naslednji korak. Ta korak je pritisk gumba, s katerim bomo izvajali interaktivno dejanje. Dejanje mora obvezno aktivirati eno ali več modifikacijskih tipk, kar ta način naredi primeren za na primer premikanje okna ali spreminjanje njegove velikosti. Če bi to delali brez Easystroka, bi bilo treba ves čas držati tipko <Alt> na tipkovnici. Ko z gesto aktiviramo tipko <Alt>, ta ostane aktivna, dokler ne spustimo gumba, torej nam v vmesnem času ni treba več skrbeti zanjo. Odvisno od gumba, ki ga pritisnemo, lahko z isto gesto začnemo več operacij, ki potrebujejo iste modifikacijske tipke (na primer levi gumb za premikanje, srednji gumb za spreminjanje velikosti), kar spet zmanjša kognitivno obremenitev uporabnika.

Treba je predstaviti še preostala dva tipa ukaza – ukazni stavek in vpis besedila. To sta najenostavnejša tipa ukazov. Ukazni stavki se najpogosteje uporabljajo za zagon programa oziroma aplikacije. V polju *Details* vpišemo stavek, ki bo ob uporabi geste pognal neki program. *Vpis besedila* uporabimo, kadar velikokrat vpisujemo enak kos besedila, recimo datum. Besedilo, ki ga nastavimo pod dejanje, se bo avtomatično vneslo v polje dokumenta ali ukazne vrstice vsakič, ko uporabimo ustrezno gesto.

6 GESTE V WINDOWSIH

Eno izmed najboljših orodij za interpretacijo gest in izvajanje ustreznih dejanj za operacijski sistem Microsoft Windows je program StrokeIt. Zadnja različica programa v trenutku pisanja tega besedila je 0.9.7; za prikaz možnosti in načina uporabe orodja bomo uporabljali različico 0.9.7 Home. Orodje ne potrebuje nobenih dodatnih knjižnic in deluje v vseh različicah operacijskega sistema Windows od Windows 95 naprej. StrokeIt je popolnoma brezplačen za osebno neprofitno uporabo [11].

6.1 StrokeIt

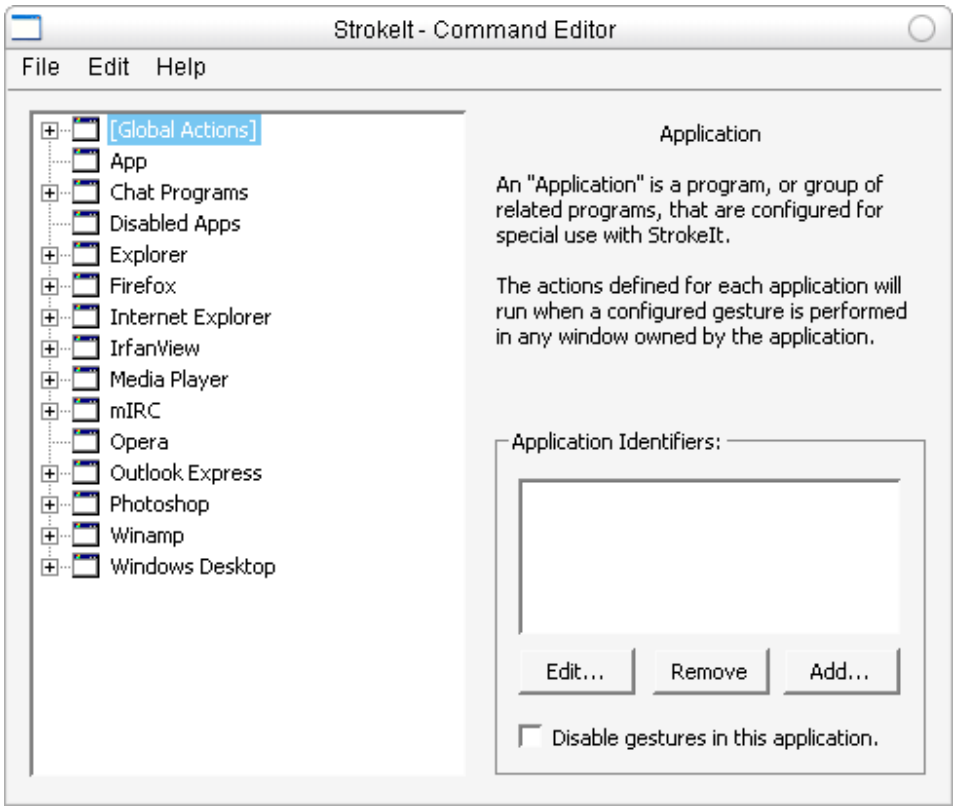
Orodje StrokeIt je zelo majhen program. Na trdem disku zavzame okrog 316 KB, pomnilniški odtis je celo manjši od tistega, ki ga ima Easystroke – le 90–300 KB. Ne potrebuje nobene dodatne knjižnice in podpira vtičnike. Če ga še malo primerjamo z Easystrokom, lahko ugotovimo nekaj bistvenih tehničnih razlik. Easystroke vsebuje algoritem s popolnim učenjem, zanaša se na uporabnika, da ga "nauči" vsako gesto. Geste obravnava vizualno s pomočjo primerjave slik. Zato zadostuje, da le enkrat narišemo gesto in Easystroke si jo zapomni. Ker zelo dobro prepoznava in fleksibilno obravnava geste, ne pride v paketu s prednameščeno začetno bazo privzetih gest. Vse geste doda uporabnik sam. Ravno nasprotno pa je v primeru programa StrokeIt, ki ne obravnava geste na enak način in je zelo nagnjen k napakam pri prepoznavanju gest. Ko vnašamo lastne, izmišljene geste, le redko zadostuje enkratno risanje geste. Treba jo je narisati večkrat, in sicer hitro ter "nerodno". Treba je vnesti več različic geste, za katero pričakujemo, da jo bomo risali v prihodnosti. Če tega ne storimo, bomo morali vedno paziti, da rišemo gesto zelo podobno tisti, ki smo jo določili prvič, sicer se lahko zgodi, da StrokeIt geste ne prepozna, ker ima na primer en krak prekratek ali pa je preveč ukrivljena. Zaradi teh težav je priporočljivo vsakemu dejanju pridružiti eno od privzetih gest v bazi, ki prihaja s programom. Privzetih gest je nekaj več kot 80 in so v glavnem v obliki številk, črk oziroma so enostavne poteze, kot so ravne ter poševne črte in podobno. Seznam privzetih gest najdemo na StrokeIt domačem spletnem mestu [12]. Poleg navedenih razlik tehnične narave obstajajo bistvene razlike v značilnostih obeh orodij. StrokeIt podpira samo prvo vrsto naprednih gest. Nima možnost izbire dodatnih sprožilnih gumbov, osnovnega sprožilnega gumba pa ni mogoče uporabljati v

kombinaciji z modifikacijskimi tipkami. Prav tako ni mogoče dostopati do dnevnika uporabljenih gest, ker ni implementiran. StrokeIt ima vseeno nekaj pomembnih podobnosti z Easystrokom. Lahko določimo aplikacije, znotraj katerih bodo geste onemogočene. Določene aplikacije imajo lahko svojo "lastno različico" geste, torej ima lahko neka gesta znotraj ene aplikacije drugačno dejanje kot znotraj drugih. Gestam lahko določimo časovno omejenost oziroma *timeout*.

6.2 Zgled programskega vmesnika

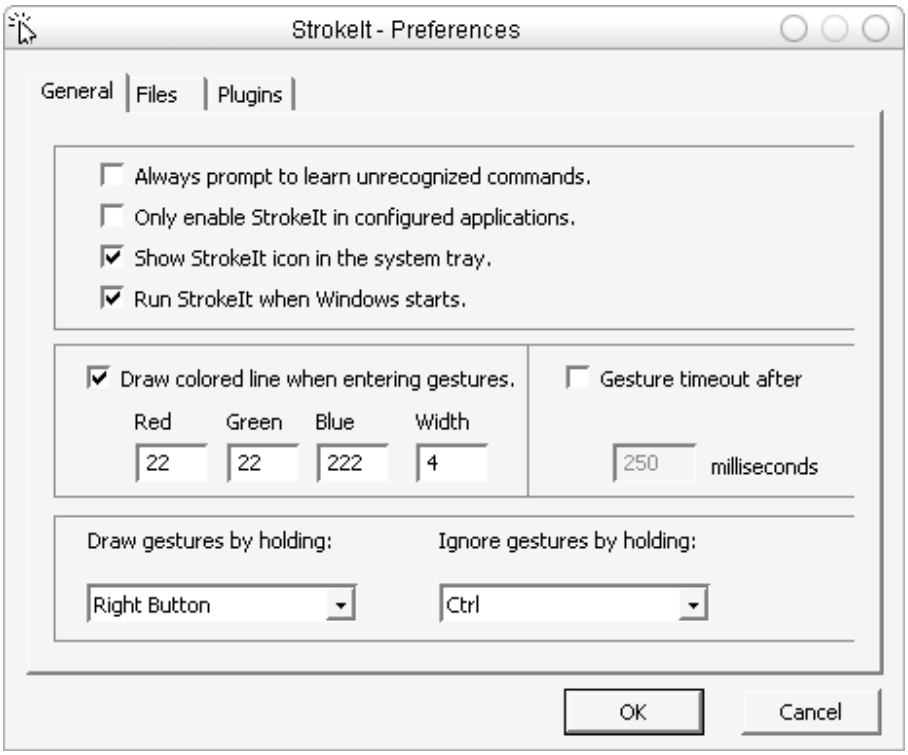
Programska ikona, ki leži v vrstici za obvestila, ima tri funkcije. Služi kot indikator stanja, v katerem je trenutno program. Če je bele barve, program deluje normalno in čaka na geste. Modra barva pomeni, da rišemo gesto. Rumena ikona pomeni, da je za naslednji klik sprožilnega gumba risanje gest onemogočeno; StrokeIt je v stanju mirovanja, dokler ne kliknemo sprožilnega gumba. Rdeča barva pomeni, da je StrokeIt onemogočen in sprožilni gumb ne bo sprožil risanja gest. Druga funkcija je nadzor dejavnosti programa. StrokeIt lahko začasno zaustavimo oziroma onemogočimo z desnim klikom na ikono. Ikona bo v tem primeru rdeča. Tretja funkcija je dostop do glavnega nastavitvenega okna (slika 26). Odpremo ga z levim klikom na ikono.

Nastavitveno okno je razdeljeno na dva dela: levo je drevesni prikaz nameščenih dejanj, desno pa podrobnosti za vsako komponento drevesa. Dejanja so v drevesu razdeljena znotraj skupin aplikacij, kar je nekoliko drugačen pristop od Easystrokovega. Če hočemo spremeniti dejanje v eni skupini, moramo spet ustvariti dejanje s spremenjenimi lastnostmi ali ga kopirati iz globalne skupine in mu nato spremeniti lastnosti. Novo narejene skupine aplikacij ne podedujejo seznama globalnih dejanj kot v Easystroku. Namesto tega moramo spet določiti dejanja, ki jih hočemo spremeniti. Nove skupine aplikacij se dodajajo s *File – New Application*, dejanja znotraj teh pa s *File – New Action*. Dejanju pridružimo ukaz s *File – New Command*. Vsakemu dejanju lahko pridružimo več ukazov, kar je nekoliko boljše kot v Easystroku. Poleg več ukazov ima lahko dejanje več pridruženih gest. Program lahko popolnoma izklopimo s *File – Shutdown StrokeIt*. Če hočemo prihraniti lastno konfiguracijo StrokeIta in jo morda uporabiti na drugem računalniku, lahko podatke izvozimo s *File – Export Config*. Konfiguracija za izvoz je razdeljena na dva neodvisna seznama dejanj in gest; s kljukicami označimo nastavitve, ki jih želimo izvoziti in kliknemo gumb *Export*. Konfiguracija se shrani v datoteki s končnico .sxp. To potrebujemo pri uvozu konfiguracije, kar dosežemo s *File – Import Config*.



Slika 26: Nastavitveno okno programa StrokeIt

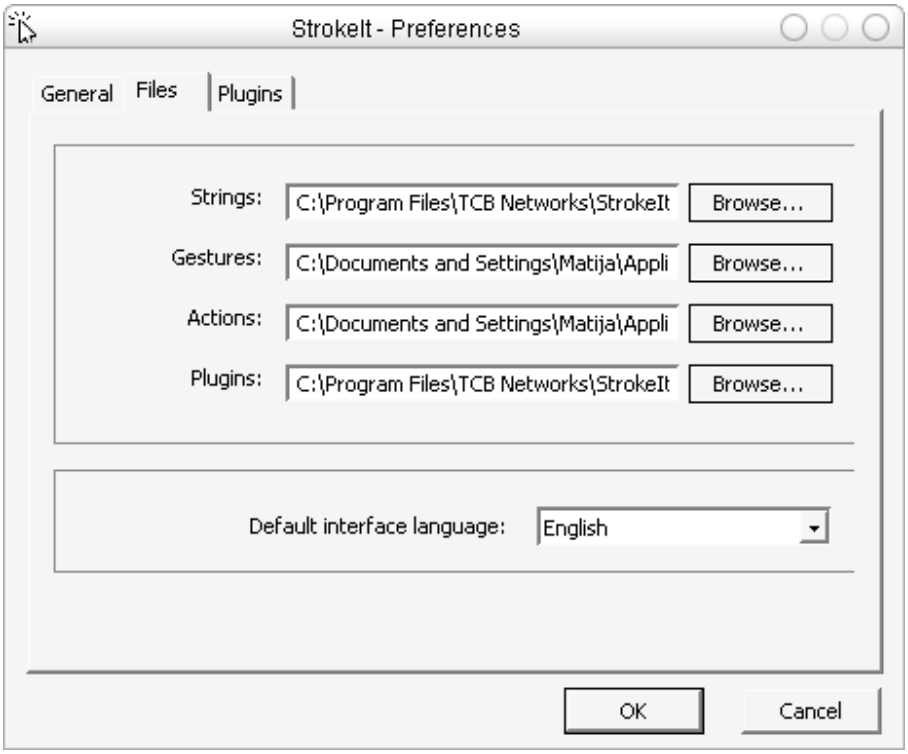
Okno z izbirnimi nastavitvami odpremo s *File – Preferences*. V prvem zavihku okna najdemo splošne lastnosti StrokeIta (slika 27). Možnost *Always prompt to learn unrecognized commands* bo poskrbela za odpiranje okna za učenje novih gest vsakič, ko se zgodi, da StrokeIt ne prepozna geste. *Only enable StrokeIt in configured applications* bo dovolilo uporabo gest samo znotraj tistih aplikacij, ki so posebej določene v seznamu aplikacij v glavnem oknu. Če ne želimo, da kazalček pušča sled v času risanja gest, lahko odkljukamo nastavitev *Draw colored line when entering gestures*. Sicer lahko le spremenimo videz sledi z vnosom vrednosti za barvo (*Red, Green, Blue*) in nastavimo širino (*Width*). *Gesture timeout* določa časovno omejitev geste. Gesto je mogoče razveljaviti s klikom levega gumba miške, preden spustimo sprožilni gumb. Sprožilni gumb lahko izberemo v spustnem meniju *Draw gestures by holding*. Na izbiro so le trije gumbi miške, brez možnih modifikacijskih tipk. Te so uporabljene v namen ignoriranja geste, ki smo jo začeli risati. Eno izmed njih izberemo v spustnem meniju *Ignore gestures by holding"*.



Slika 27: Izbirne nastavitve

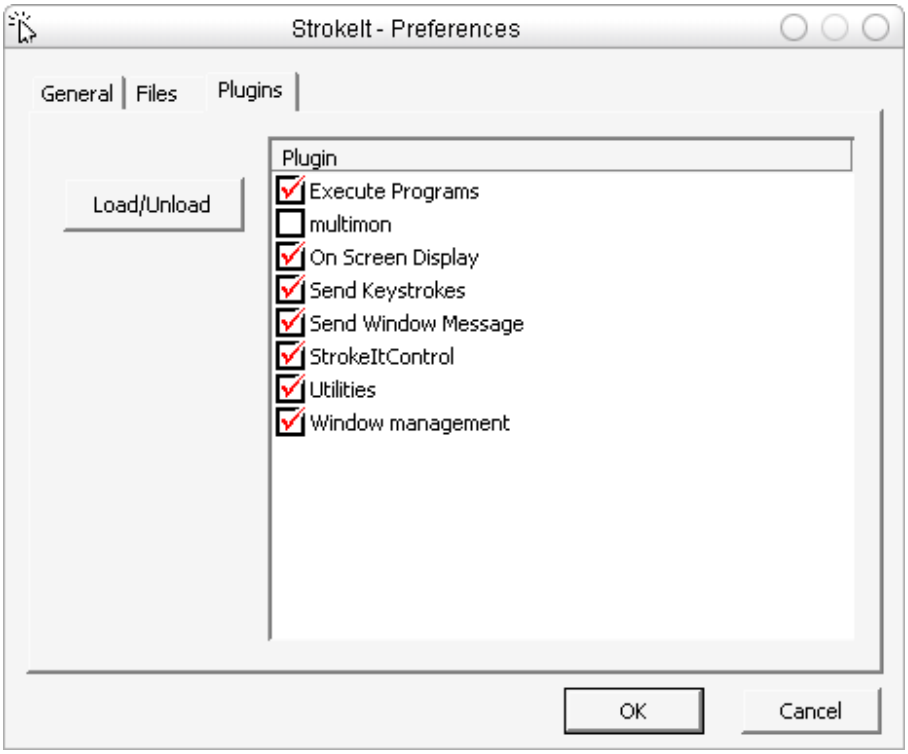
V drugem zavihku imamo možnost izbire poti do posebnih datotek programa StrokeIt (slika 28). *Strings* označuje položaj datotek, ki vsebujejo prevode programskega vmesnika. V polju *Gestures* je naslov datoteke, v kateri so shranjene geste, v *Actions* pa naslov, na katerem najdemo datoteke s skriptami za vsako posamezno dejanje. Ukazi, ki se izvedejo ob dejanju, so shranjeni v ustrezni skriptni datoteki. *Plugins* vsebuje naslov, kjer so vtičniki. Spustni meni *Default interface language* ponuja na izbiro jezik, v katerem bo prikazan programski vmesnik. Določeni jezik je dostopen, ko so ustrezne datoteke s prevodi na položaju, določenem v polju *Strings*.

Možnost izbire datotek programa nam omogoča, da programske nastavitve začasno zamenjamo brez spremembe glavnih nastavitev, ki se uporabljajo na računalniku. Npr. če hočemo začasno uporabljati naše geste in dejanja na prijateljevem računalniku, lahko usmerimo StrokeIt na ustrezne datoteke na naši USB pomnilniški napravi. S tem se izognemo večjim in dolgoročnim spremembam, kot so izvoz in uvoz konfiguracije ali nameščanje novega uporabnika v operacijskem sistemu.



Slika 28: Izbira konfiguracijskih datotek

Tretji zavihek prikazuje dostopne vtičnike, ki so na mestu, vnesenem v polju *Plugins* v predhodnem zavihku. Z gumbom *Load/Unload* naložimo ali sprostimo posamezne vtičnike. Naloženi so označeni s kljukico.



Slika 29: Izbira vtičnikov

6.3 Nastavljanje dejanj

Ko prvič zaženemo StrokeIt, ima ta že veliko nameščenih dejanj, ki so priročna za nove uporabnike in služijo kot primeri. V našem primeru bomo začeli s praznim seznamom. Za primer nameščanja dejanj bomo uporabili geste, ki so čim bolj podobne tistim, ki smo jih že uporabljali v Easystroku. S tem bomo videli, koliko je možno rekonstruirati enak občutek vmesnika, na katerega smo že navajeni, vendar v popolnoma drugačnem operacijskem sistemu in z uporabo drugega orodja.

StrokeIt podpira več vrst ukazov, ki so razdeljeni po skupinah. Med "Internet" ukaze spadajo taki, kot sta ustvarjanje elektronske pošte in odpiranje spletnih naslovov. "Keys" ukazi so tisti, ki sprožijo eno ali več tipk s tipkovnice. Pošilja se lahko bližnjice, zaporedne sprožitve določenih tipk ali skrita gesla v ciljno aplikacijo. "StrokeIt" ukazi sprožijo dejanja v okviru samega orodja:

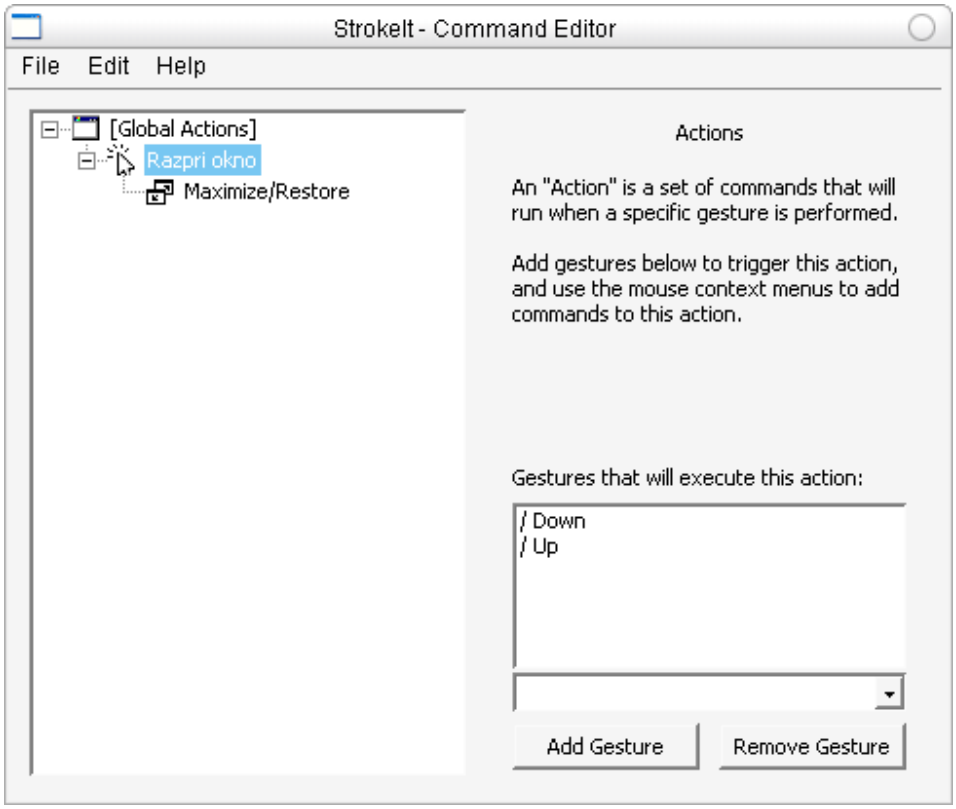
prekinejo, začasno onemogočijo ali dajo izvajanje programa v mirovanje, odprejo nastavitveno okno in odprejo okno za učenje gest. "Utilities" so ukazi za prikazovanje in potrditev obvestilnih dialogov, predvajanje zvočnih opozoril, prestavitev zaslona v stanje spanja ... "Windows" ukazi so tisti, ki služijo za upravljanje z okni. Dostopne možnosti so: aktiviranje okna, razpiranje ali vrnitev na prejšnjo velikost, skrčitev okna, premik na določene koordinate in sprememba na določeno velikost. Razvidno je, da manjkajo funkcionalnosti, kot so premikanje okna v kote namizja, ročno premikanje, ročno spreminjanje velikosti in še nekatere zelo uporabne možnosti, ki smo jih lahko dosegli z Easystrokom in Metacityem v GNU/Linuxu. Kako bomo dodali takšne funkcionalnosti bo pojasnjeno kasneje. "Run" je vrsta ukaza za poganjanje zunanjih programov. Nastavljiva je tako datoteka programa kot tudi parametri ukazne vrstice. "WinMsg" so ukazi za pošiljanje posebnih sporočil znotraj Windows GUV-a. S pomočjo teh sporočil lahko upravljamo vse funkcije okna na sistemski ravni.

Možnost izvajanja več ukazov v enem dejanju nam omogoča na primer, da aktiviramo določeno okno in mu hkrati pošljemo bližnjico oziroma kakšen kos besedila. Nekateri ukazi so namenjeni prav izvajanju v kombinaciji z drugimi. "On screen display" je ukaz, ki izpiše zaslonsko obvestilo. Vsebino obvestila lahko določimo sami. Te ukaze je pametno uporabljati skupaj z glavnimi, tako da nas obvestijo, katero dejanje smo sprožili. S tem bomo dobili koristno vizualno povratno informacijo enako tisti, ki jo omogoča nastavitev "Show popups" v Easystroku. Večino ukazov vrste "Utilities" je logično uporabljati v kombinaciji z drugimi. "Delay" povzroči čakanje za določen čas, preden se sproži glavni ukaz. "Messagebox" služi prikazovanju sporočilnega okna, "Play Sound" pa predvajanju zvočnega opozorila hkrati z izvajanjem glavnega ukaza. "Ok/Cancel Messagebox" je sporočilni dialog z gumboma *Ok* in *Cancel*, s katerima lahko nadaljujemo ali preprečimo izvajanje glavnega ukaza. Tak dialog lahko uporabimo na primer kot varnostno vprašanje, preden se izvede kakšen morebitno nevaren ukaz.

Pri izbiranju gest bomo tudi tokrat uporabljali tri načela za določanje kot pri Easystroku (enostavnost, intuitivnost, različnost). Kot je že omenjeno, StrokeIt ne podpira naprednih gest, razen prve vrste. To pomeni, da lahko poleg sprožilnega gumba uporabimo enega od drugih gumbov le, če se miška ne premika. Geste so pri StrokeItu večinoma prav to, kar so: narisani simboli, ki se med seboj razlikujejo le po obliki risbe. Klikanje vmesnih gumbov ne pride v poštev kot del risane geste, zato gumbov ne moremo uporabljati kot elemente, ki pri gestah naredijo razliko na nevizualen način. Razlike med vizualnimi gestami so torej lahko le vizualne. Za geste z enako orientacijo moramo poskrbeti, da se čim bolj vizualno razlikujejo, vendar čim

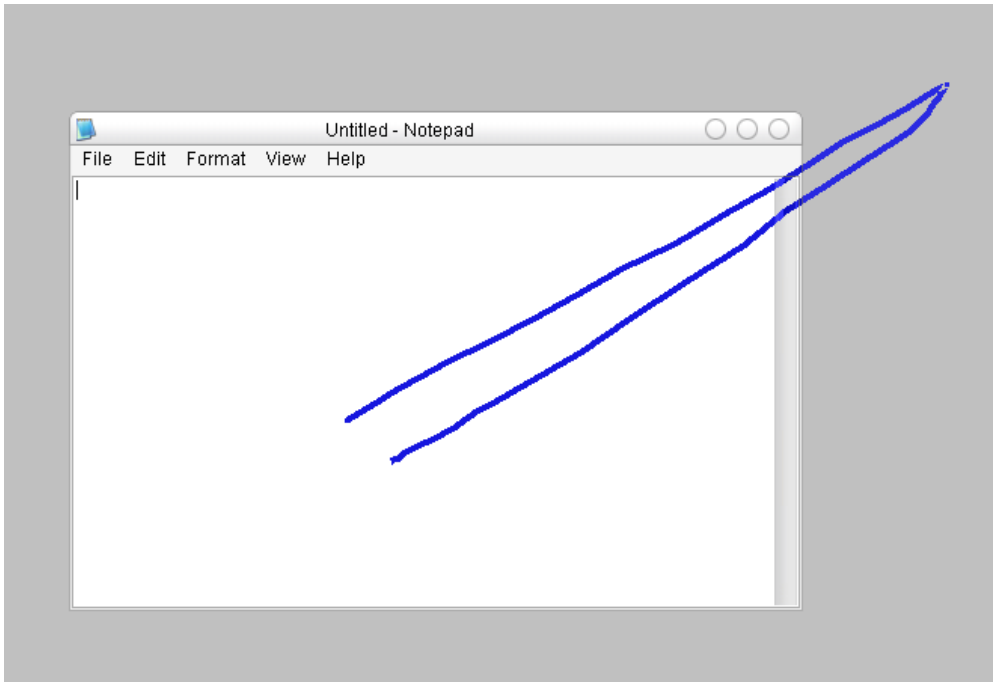
manj na račun enostavnosti, ker sicer zmanjšujemo intuitivnost. Nekaj takih problemov bomo poskusili rešiti v naslednjih primerih.

Ustvarili bomo novo dejanje, ki nam bo razpiralo okno. V skupini *[Global Actions]* dodamo novo dejanje in ga ustrezno poimenujemo. Potem ga označimo in na desni strani okna se pojavi spustni meni za izbiro geste. Recimo, da smo že navajeni na Easystroke v GNU/Linuxu in hočemo imeti tudi v Windowsih čim bolj podobne geste za enaka dejanja. V spustnem meniju izberemo "/ Up", kar pomeni poševno črto proti zgornjemu desnemu kotu namizja. Kliknemo gumb *Add Gesture* in gesta se doda na seznam. Nato dodamo nov ukaz za naše dejanje. V spustnem meniju za ukaze izberemo *Windows – Maximize or Restore*. Ker sta oba ukaza, torej razpiranje in razveljavitev razprtosti (vračanje prejšnje velikosti), združena znotraj enega ukaza, bomo dejanju dodali še eno gesto v obliki poševne črte proti levem spodnjem kotu namizja – to je gesta "/ Down". Slednjo smo imeli v Easystroku za razveljavitev razprtosti, kar je bilo ločeno dejanje. Uporabili jo bomo tudi tukaj, da se izognemo morebitni zmedi.



Slika 30: Namestitev novega dejanja

Naslednje dejanje bo krčenje okna. Gesta bo spet enaka kot v Easystroku, navpična črta navzdol oziroma "Down". Za ukaz izberemo *Windows – Minimize*. Naslednja dejanja, ki jih bomo ustvarili, bodo premikala okna v kote in središče namizja. Težave nastanejo pri izbiri geste za premik okna v zgornji desni in spodnji levi kot. Prvo dejanje ima enako orientacijo kot že nastavljena gesta za razpiranje okna, drugo pa enako orientacijo kot gesta za povrnitev prejšnje velikosti okna. Takšne probleme smo v Easystroku reševali z uporabo naprednih gest, kjer je en dodatni klik zagotovil vso potrebno razliko med gestami, brez spreminjanja risbe simbola. StrokeIt nima ustreznih tipov naprednih gest, ki bi lahko rešili ta problem. Želimo še naprej uporabljati že nastavljene geste, na katere smo navajeni, poleg tega pa želimo tudi podobne risbe za premike oken v kote namizja. Risbe bomo morali res nekoliko spremeniti, vendar tako, da obdržijo čim večjo enostavnost in intuitivnost tudi v tem primeru. Namesto prejšnjega klika gumba bomo gestam dodali potezo v nasprotni smeri. Premik okna v zgornji desni kot sprožimo v dveh potezah – poševna črta v smeri zgoraj desno in poševna črta v smeri spodaj levo ("/ Up – / Down"), kot prikazuje slika 31. Podobno bomo naredili za premik okna v spodnji levi kot – poševna črta v smeri spodaj levo, ki ji sledi poševna črta v smeri zgoraj desno ("/ Down – / Up").



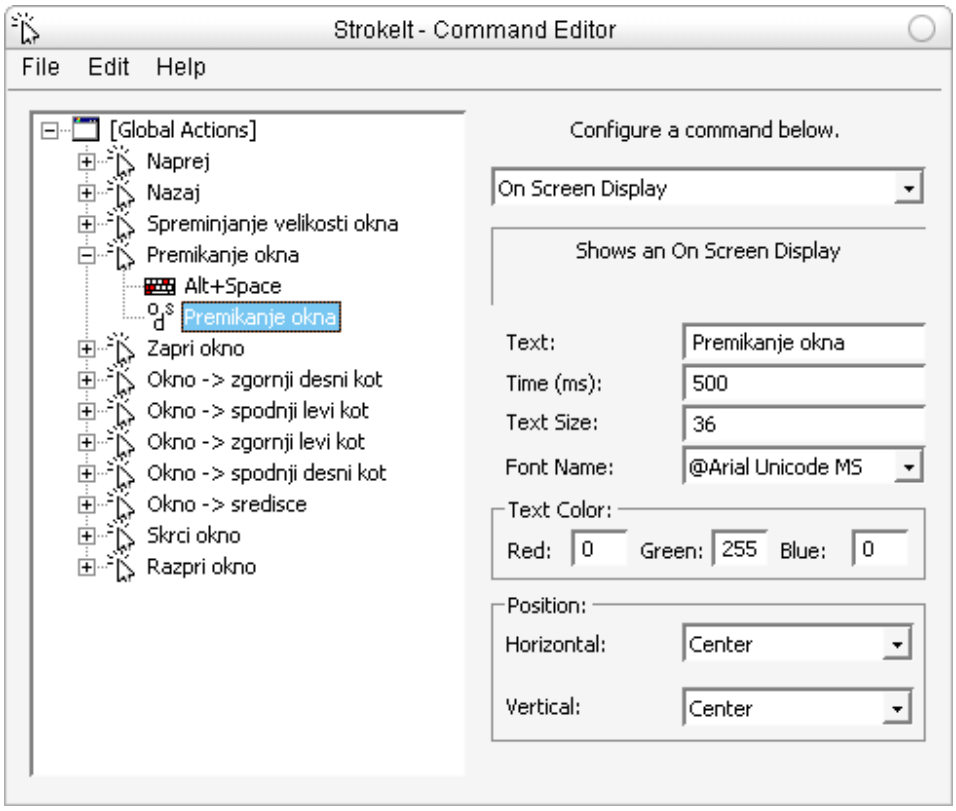
Slika 31: Gesta za premik okna v zgornji desni kot

Podobne geste z ustreznimi orientacijami bomo nastavili za premike v preostale kote namizja. Dejanju, ki premakne okno v središče namizja, bomo na enak način pridružili gesto "dol-gor" ("Down-Up"), da ne bi prišlo do konflikta z gesto "dol", ki je pridružena dejanju za krčenje okna. Dejanj za premikanje okna, ki smo jih pravkar dodali, ne moremo doseči z vgrajenimi ukazi v StrokeIt. Te funkcionalnosti bomo pustili za kasneje.

Nadaljujemo z dejanji za pomikanje po zgodovini brskanja nazaj in naprej. Ukaza bosta izkoriščala pošiljanje bližnjic v okno. Izberemo *Keys – Hotkey* in v desnem delu nastavitvenega okna vtipkamo bližnjico <Alt> + <Right> za naprej oziroma <Alt> + <Left> za nazaj. Dejanje za zapiranje okna bo uporabljalo bližnjico <Alt> + <F4>. Za gesto bomo izbrali črko "K", ker je njeno risanje zelo podobno gesti, ki smo je imeli v Easystroku.

Slika 32: Oblika geste "K"

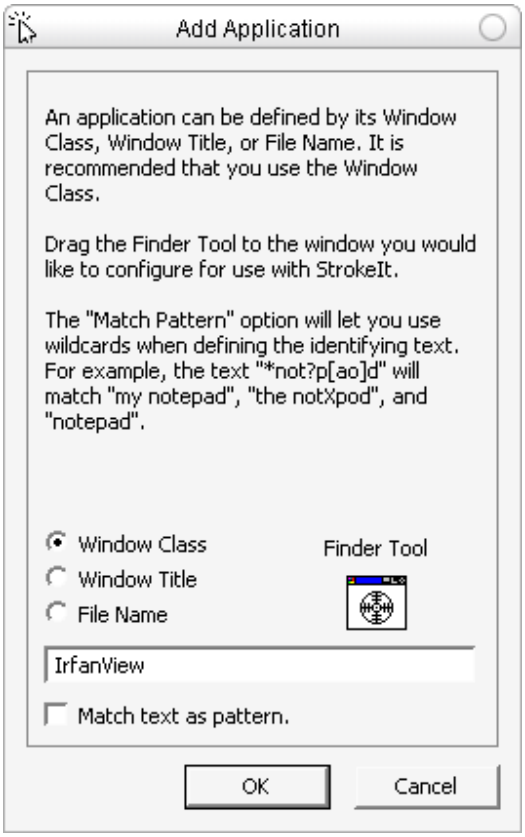
Za naslednji primer bomo naredili dve dejanji, ki uporabljata geste brez risanja, uporabljata dodaten gumb in spadata v prvi tip napredne geste pri Easystroku. Prvo dejanje bo služilo za ročno premikanje okna po namizju. Kot gesto izberemo "LBUTTON_DOWN", kar pomeni, da je gumb v pritisnjenem stanju. To je pomembno, ker bo dejanje aktivno, dokler držimo levi gumb (tudi če spustimo sprožilnega), kar nam bo omogočalo, da s premikanjem kazalčka premikamo tudi celotno okno. Gesto izvedemo tako, da najprej pritisnemo sprožilni gumb in med držanjem pritisnemo levi gumb miške. Med dvema pritiskoma miške ne premikamo. Funkcionalnost bomo temu dejanju dodelili kasneje. Za zdaj lahko dodamo bližnjico <Alt> + <Space>, ki jo bomo pozneje uporabljali za to dejanje, in ukaz "On Screen Display". Ta ukaz nas bo obveščal, da smo sprožili premikanje okna. V polje *Text* vpišemo besedilo "Premikanje okna". Potem nastavimo (v milisekundah), koliko časa bo obvestilo prikazano na zaslonu, ter izberemo velikost (*Text Size*) in tip pisave (*Font Name*). Če želimo, lahko izberemo še barvo pisave (*Text Color*) in položaj na zaslonu (*Position*).



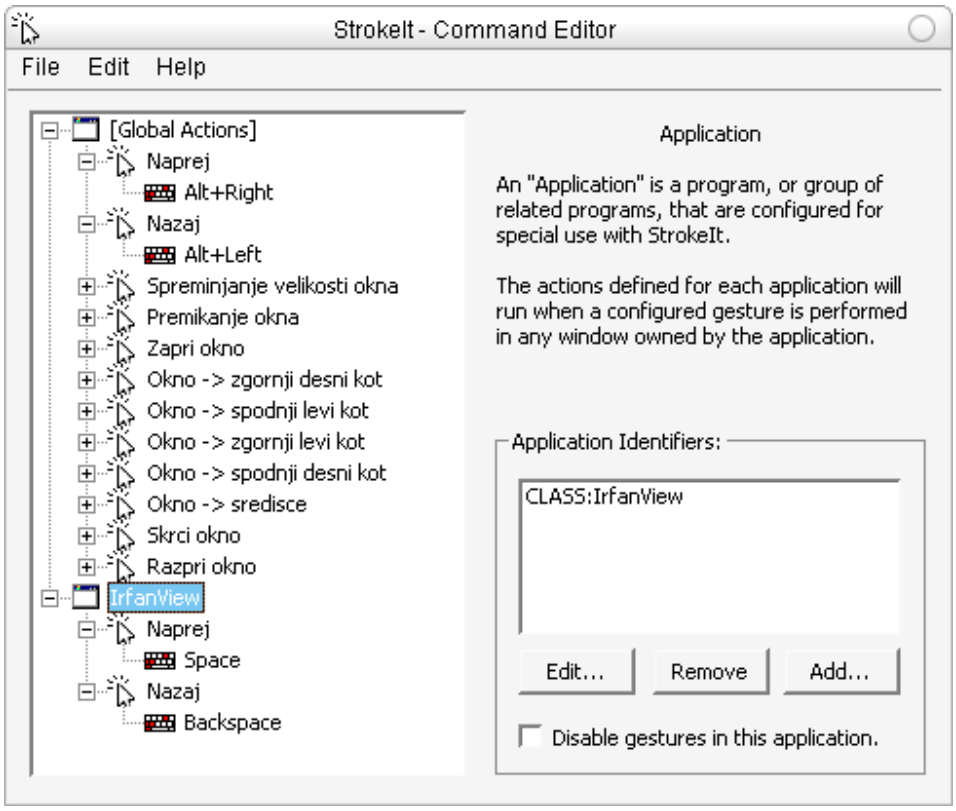
Slika 33: Urejanje zaslonskega obvestila

Gesto za ročno spreminjanje velikosti okna bomo ustvarili podobno prejšnji, razlika bo le v izbiri "MBUTTON_DOWN". Iz spustnega menija za izbiro gest je razvidno, da lahko uporabljamo obe stanji (gumb pritisnjen in gumb spuščen) za naslednje gumbe miške: miškin kolešček, levi gumb, srednji gumb. Za desni gumb lahko v našem primeru uporabimo le pritisnjeno stanje, ker se spuščeno stanje uporablja za prekinitev gest (desni gumb je sprožilni). Z naslednjim primerom bomo pokazali spremembo že obstoječih dejanj znotraj določene aplikacije. Primer bo aplikacija IrfanView, ki uporablja drugačni bližnjici za sprehod naprej/nazaj. V drevo dejanj na levi strani nastavitvenega okna bomo dodali novo skupino dejanj, ki jo bomo poimenovali "IrfanView". Skupino označimo, v desnem delu okna pa se prikaže seznam identifikatorjev oken, znotraj katerih bo delovala skupina dejanj. Na seznam bomo dodali osnovno okno aplikacije IrfanView. Klik gumba *Add* odpre okno za vpis identifikatorja (slika 34). Kot identifikator lahko uporabimo razred okna (*Window Class*), naslov okna (*Window Title*) ali naziv datoteke programa (*File Name*). Drugi, morda lažji način je z uporabo majhnega vgrajenega orodja *Finder Tool*. Majhno

ikono primemo in odvlečemo na odprto okno aplikacije IrfanView na namizju, da orodje ugotovi ustrezen identifikator. S klikom gumba *Ok* potrdimo izbiro in okno je dodano na seznam identifikatorjev. Ko končamo identificiranje, iz skupine *[Global Actions]* kopiramo dejanje za naprej in dejanje za nazaj (*Edit – Copy*) ter ju prilepimo v našo novo skupino (*Edit – Paste*). S tem se izognemu ponovnemu ustvarjanju dejanj, saj jima moramo le še spremeniti ustrezni bližnjici. Bližnjica za nazaj v IrfanViewu je tipka <Backspace>, za naprej pa tipka <Space>. Od zdaj naprej lahko isti gesti za naprej in nazaj uporabljamo v brskalniku datotek, spletnem brskalniku in aplikaciji IrfanView (slika 35).



Slika 34: Okno za vnos identifikatorja

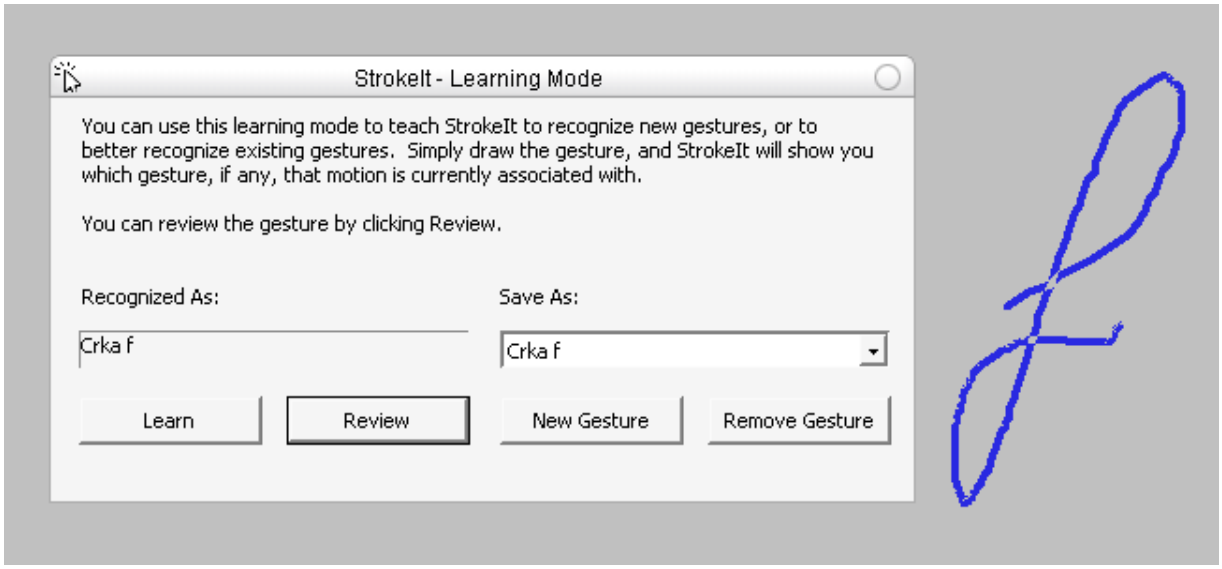


Slika 35: Spremembe dejanj za aplikacijo IrfanView

Zanimiva možnost je tudi ta, da lahko skupine, ki sicer služijo za spremenjeno delovanj gest znotraj nekaterih aplikacij, uporabimo kot množice aplikacij, v katerih StrokeIt sploh ne bo deloval. To naredimo tako, da ustvarimo novo skupino posebej za prepovedane aplikacije. Te je treba le dodati na seznam s pomočjo identifikatorjev in odkljukati nastavitev *Disable gestures in this application*. Na seznam lahko dodamo več identifikatorjev. To možnost uporabljamo za aplikacije, znotraj katerih lahko dejanja, ki jih sprožijo geste StrokeIta, povzročijo nepredvidene posledice. To so po navadi večinoma igre, lahko pa so tudi aplikacije z lastnimi vgrajenimi gestami, na primer spletni brskalnik Opera.

Za konec bomo ustvarili prilagojeno novo gesto, ki je v StrokeItu ni. Do zdaj smo uporabljali vgrajene geste, ker so zelo zanesljive in jih StrokeIt večinoma prepozna brez problemov. Ročno ustvarjene geste v StrokeItu imajo to težavo, da jih program težko prepoznava ali razlikuje, razen če niso zelo natančno narisane oziroma če ga nismo zelo dobro "naučili" brati naše risbe. Zato je

isto gesto treba vnašati večkrat, hitro in na "nespreten" način, tako, kakor bi jih sicer risali v vsakdanji uporabi in v vseh priložnostih. Ustvarili bomo gesto v obliki pisane črke "f", ki jo bomo kasneje lahko uporabili na primer za odpiranje spletnega brskalnika Firefox. Odpremo okno za učenje gest (*Edit – Learn Gestures*). Kliknemo gumb *New Gesture* in vpišemo ime naše nove geste, na primer "Črka f". Potem jo najdemo in izberemo na seznamu v spustnem meniju *Save As*. Našo novo gesto prvič narišemo. V polju *Recognized As* bo pisalo "<Unrecognized Gesture>", kar pomeni, da StrokeIt ni prepoznal narisane geste kot ene izmed shranjenih. Našo risbo si bo StrokeIt zapomnil kot gesto z imenom "Črka f", potem ko kliknemo gumb "Learn". S tem delo še ni končano, saj je treba še čim večkrat ponavljati risanje in če pride do "<Unrecognized Gesture>", vsakič znova klikniti gumb *Learn*. To bo zagotovilo, da si bo StrokeIt zapomnil vse naše risbe z vsemi minimalnimi razlikami in bo te razlike v prihodnje spregledal. Na tak način bomo "naučili" algoritem, da se manjkrat zmoti v primerih, ko je en krak naše geste predolg ali kadar je, recimo, gesta popačena. Ko se zgodi, da nismo prepričani, da risba dovolj dobro ponazarja našo črko "f", lahko preverimo, kaj smo nazadnje narisali, s klikom gumba *Review*.



Slika 36: Učenje gest

GNU/Linux nam podarja možnosti in funkcionalnosti, ki nam niso dostopne v Windowsih. Razvidno je, da ima Metacity kot eden od najbolj enostavnih upravljalnikov oken v svetu

GNU/Linux že vgrajene funkcionalnosti, ki razširjajo dosedanji tradicionalni GUV, torej tistega, ki ga je svet prvič spoznal pri računalniku Apple II. Takšne razširitve lahko postopoma spremenijo osnovne navade uporabnika in izpopolnijo občutek modernega, intuitivnega in zelo učinkovitega grafičnega vmesnika, ki je usmerjen predvsem k čim hitrejšemu reševanju problemov in čim prejšnjemu doseganju rezultatov. Pokazali smo, kako uporabljati skrite funkcionalnosti Metacityja s pomočjo bližnjic, da pospešimo svoje delo. Te funkcionalnosti smo povezali z Easystrokom in jih pripeljali na višjo raven: intuitivne geste so zamenjale morda prezapleteno uporabo bližnjic in našo pozornost preusmerile le na miško. Že enostavna dejanja, podprta v upravljalniku oken GNU/Linuxa, kot so ročno premikanje in spreminjanje velikosti okna s kazalčkom kjerkoli na oknu, smo dodatno poenostavili s pomočjo Easystrokovih naprednih gest. Z drugimi besedami, Metacity nam je omogočil koristne možnosti, ki jih Windows nima, Easystroke pa je prenesel njihovo upravljanje na miško samo. Znotraj okolja Windows takšna možnost ni lahko dostopna, StrokeIt je ne podpira. StrokeIt na primer omogoča premik okna na določene koordinate namizja (ukaz *Windows – Move*), ampak to ne zagotavlja premika okna v, recimo, zgornji desni kot, ker so v tem primeru ustrezne koordinate odvisne od trenutnih dimenzij okna. Vse te pomanjkljivosti Windowsa in StrokeIta rešimo z uporabo orodja AutoIt.

7 AUTOIT

AutoIt je brezplačen zaprtokodni avtomatizacijski skriptni jezik za Microsoft Windows. Pojavil se je konec leta 1998 kot program v C-ju, narejen za avtomatizacijo vnosov tipkovnice v času namestitve programske opreme. Od tedaj je šel AutoIt skozi številne spremembe in posodobitve, tako da je do danes zrasel v bogat in prilagodljiv programski jezik tretje generacije s sintakso BASIC in klasičnim podatkovnim modelom. Glavni namen AutoIta je ustvarjanje skript, ki lahko avtomatizirajo skoraj vse v okolju Windows, od nalog, kot so namestitev aplikacije ali določene spremembe nastavitev, do nadzora vhoda miške in tipkovnice. Avtomatizacija je veliko bolj poenostavljena in zanesljiva kot z uporabo nekaterih drugih programskih jezikov. Skripte lahko služijo za opravljanje sistemskih administratorskih nalog brez nadzora in zato omogočajo tudi zvišanje oziroma razširitev ravni upravljanja samega sistema, na primer prek GUV-a. Veliko dodatne in specializirane funkcionalnosti omogočajo uporabniško narejene knjižnice, t. i. UDF-ji (angl. "User Defined Functions"), ki so dostopne tako v standardnem paketu kot na domačem spletnem mestu.

AutoIt podpira variablini podatkovni tip, v katerem lahko hranimo več tipov podatkov, vključno s tabelami. Z njim lahko ustvarimo kompleksne izraze, funkcije, zanke, objekte COM ("Component Object Model"), elemente GUV-a itd. Upravljamo lahko s standardnimi Windows ukazi, okni in procesi ter simuliramo miško in tipkovnico. AutoIt je zasnovan tako, da bo čim manjši in nezahteven. Ne potrebuje nobene zunanje DLL knjižnice in ne uporablja Windowsovega registra [13]. S pomočjo orodja Aut2Exe skripte lahko prevedemo v samostojne izvedljive datoteke, ki jih lahko poženemo tudi na drugih računalnikih, brez potrebe po nameščanju dodatnih okolij za izvajanje.

AutoIt nam bo omogočil razširitev možnosti StrokeIta in tudi nadgradnjo Windowsa z vidika upravljanja GUV-a. Ustvarili bomo funkcionalnosti, ki prej niso bile dostopne v Windowsih, in sicer tiste, ki smo jih imeli v Metacityju in Easystroku. Skripte, ki jih bomo predstavili, so načrtovane tako, da ves čas tečejo v ozadju in čakajo na ustrezno bližnjico, preden sprožijo svojo funkcijo. Takšen pristop omogoča ne samo veliko hitrejši odziv klicanih funkcij, ampak dovoli tudi uporabo skript le z uporabo bližnjic na tipkovnici, brez posredovanja StrokeIta, če si seveda to zaželi uporabnik.

7.1 Premikanje okna

Prva skripta, s katero bomo predstavili AutoIt, nam bo omogočila tisto, kar nam brez posebnih razširitev že omogoča Metacity v GNU/Linuxu, torej premikanje okna v kote ali središče namizja.

```
HotKeySet("!#{y}", "WinMoveCenter") ;bliznjica <Alt> + <Win> + <y>
HotKeySet("!#{q}", "WinMoveUL")     ;bliznjica <Alt> + <Win> + <q>
HotKeySet("!#{w}", "WinMoveUR")     ;bliznjica <Alt> + <Win> + <w>
HotKeySet("!#{a}", "WinMoveDL")     ;bliznjica <Alt> + <Win> + <a>
HotKeySet("!#{s}", "WinMoveDR")     ;bliznjica <Alt> + <Win> + <d>

Global $active_win_handle

While True
   Sleep(900)  ;spi v ozadju in caka na klice bliznjic
WEnd

Func WinMoveCenter()   ;premakne okno v sredisce namizja
   $active_win_handle = WinGetHandle("[ACTIVE]")   ;poisce ID rocaj (oznako)
                                                   ;aktivnega okna
   Dim $winsize = WinGetPos($active_win_handle)
   Dim $x_coord = (@DesktopWidth - $winsize[2]) / 2
   Dim $y_coord = (@DesktopHeight - $winsize[3]) / 2
   WinMove($active_win_handle, "", $x_coord, $y_coord)
EndFunc

Func WinMoveUL()   ;premakne okno v zgornji levi kot namizja
   $active_win_handle = WinGetHandle("[ACTIVE]")
   WinMove($active_win_handle, "", 0, 0)
EndFunc

Func WinMoveUR()   ;premakne okno v zgornji desni kot namizja
   $active_win_handle = WinGetHandle("[ACTIVE]")
   Dim $winsize = WinGetPos($active_win_handle)
   WinMove($active_win_handle, "", @DesktopWidth - $winsize[2], 0)
```

```
EndFunc

Func WinMoveDL()    ;premakne okno v spodnji levi kot namizja
   $active_win_handle = WinGetHandle("[ACTIVE]")
   Dim $winsize = WinGetPos($active_win_handle)
   WinMove($active_win_handle, "", 0, @DesktopHeight - $winsize[3])
EndFunc

Func WinMoveDR()    ;premakne okno v spodnji desni kot namizja
   $active_win_handle = WinGetHandle("[ACTIVE]")
   Dim $winsize = WinGetPos($active_win_handle)
   WinMove($active_win_handle, "", @DesktopWidth - $winsize[2], @DesktopHeight -
$winsize[3])
EndFunc
```

Funkcije *HotKeySet* združijo bližnjice, ki jih bomo klicali s tipkovnico, z določeno programsko funkcijo. Tako na primer stavek

```
HotKeySet("!#{y}", "WinMoveCenter")
```

nastavi kombinacijo tipk <Alt> + <Win> + <y> kot bližnjico za klic funkcije *WinMoveCenter*. Črke in imena znotraj zavitih oklepajev {} označujejo tipke na tipkovnici. Modifikacijske tipke <Alt>, <Win>, <Ctrl> in <Shift> so v AutoItu predstavljene s simboli !, #, ^, +. Funkcija *Sleep* znotraj neskončne zanke poskrbi za to, da program spi v ozadju in čaka na bližnjice, dokler ga ne zapremo. *HotKeySet* in neskončno zanko s funkcijo *Sleep* bomo uporabljali na začetku vseh naših skript.

Glavni del kode sestavljajo funkcije *WinMoveCenter*, *WinMoveUL*, *WinMoveUR*, *WinMoveDL* in *WinMoveDR*. Vsaka premakne okno v središče ali enega od štirih kotov namizja. Sistemska funkcija *WinGetHandle* prvič ugotovi, katero okno je aktivno (v fokusu) v trenutku klicanja bližnjice. *WinGetPos* vrne položaj okna na namizju in njegove dimenzije. Dimenzije so zelo pomembne, ker je od njih odvisno, na katere koordinate bo okno prestavljeno, če želimo na primer precizno centrirati okno v središču namizja. Koordinate novega položaja se izračunajo odvisno od dimenzij okna in dimenzij namizja. Dimenzije niso potrebne samo, kadar okno

premikamo v zgornji levi kot namizja, saj sta takrat koordinati okna ves čas isti (0, 0). Funkcija *WinMove* premakne okno na nov položaj.

7.2 Krčenje in večanje okna

Naslednja skripta nam bo omogočila skrčitev (Angl. minimize) in povečanje okna nazaj v prejšnjo velikost (Angl. unminimize), vendar bo to izvajala tako, da bo okna večala v obrnjenem vrstnem redu od krčenja, tako kot je deloval ukaz *Unminimize* v Easystroku. Ko skrčimo vsa okna in jih potem vsa spet povečamo s funkcijo *WinUnminimize*, bodo spet, kot na začetku, vsa na istih položajih v navideznem kupu oken na namizju. To za vsako okno pomeni, da bo spet nad istimi okni kot prej in spet pod istimi okni kot prej. Krčenje in večanje oken na tak način nam olajšata in pospešita delo.

Primer: v fokusu je okno A in prekriva okno B, ki ga potrebujemo in do katerega želimo hitro priti. Lažje je skrčiti okno A kot iskati B na opravilni vrstici, posebno če imamo veliko odprtih oken. Do problema pride, ko se hočemo vrniti nazaj na okno A, ki ga moramo zdaj iskati na opravilni vrstici. Ta problem reši večanje okna v obrnjenem vrstnem redu. Ko zaženemo dejanje unminimize, se okno A vedno prvo vrne nazaj.

Skripta si mora zapomniti skrčena okna, in sicer po vrstnem redu LIFO. To najlaže dosežemo tako, da skripta sama skrči okno, ki si ga mora zapomniti. Za pomnjenje oken v AutoItu implementiramo podatkovno strukturo sklad, narejen s tabelo. Vsakič, ko ga skrčimo, se okno vrine v sklad. Ko ga spet povečamo nazaj, se okno iz njega odstrani.

```
Global $active_win_handle
Global $stack
HotKeySet("^{-}", "WinMinimize")   ;bliznjica <Ctrl> + <->
HotKeySet("^+{-}", "WinUnminimize")    ;bliznjica <Ctrl> + <Shift> + <->

Stack_Create() ;naredimo novi sklad

While True
   Sleep(900)  ;spi v ozadju in caka na klice bliznjic
WEnd
```

```
Func WinMinimize()

   $active_win_handle = WinGetHandle("[ACTIVE]")    ;poisce ID rocaj (oznako)
                                                    ;aktivnega okna
   WinSetState($active_win_handle, "", @SW_MINIMIZE)    ;skrcimo okno tako da mu
                                                        ;spremenimo stanje
   Stack_Push($active_win_handle)
EndFunc

Func WinUnminimize()    ;funkcija vrne naslednje okno iz sklada
   Do
       $minimized_win_handle = Stack_Pop()
       If ($minimized_win_handle = -1) Then ExitLoop    ;ce ni vec oken v skladu
                                                        ;(Stack_Pop vrne -1),
                                                        ;potem ustavimo zanko
   Until (BitAND(WinGetState($minimized_win_handle), 16) = 16) ;dokler ne pridemo
                                                               ;do skrcenega okna
                                                               ;(state 16).
   WinSetState($minimized_win_handle, "", @SW_RESTORE) ;povecamo okno
EndFunc

Func Stack_Create()
   Global $stack[5]
   $stack[0] = 0        ;prvo mesto v tabeli vsebuje index mesta kjer se nahaja
                        ;zadnji element (ce je 0 potem je sklad prazen)
   Return $stack
EndFunc

Func Stack_IsEmpty()
   Return ($stack[0] = 0)  ; true ali false. "=" je isto kot "==" v tem primeru.
EndFunc

Func Stack_Peek()
   If Not Stack_IsEmpty() Then
       Return $stack[$stack[0]]
   Else
       Return -1
   EndIf
EndFunc
```

```
Func Stack_Pop()
   If Not Stack_IsEmpty() Then
       $element = $stack[$stack[0]]
       $stack[$stack[0]] = ""
       $stack[0] -= 1
       Return $element
   Else
       Return -1
   EndIf
EndFunc

Func Stack_Push($new_element)
   If (Ubound($stack) - 1 = $stack[0]) Then     ;ce je zadnje mesto zasedeno
       ReDim $stack[UBound($stack) + 5]         ;povecaj tabelo za novih 5 mest
   EndIf
   $stack[$stack[0] + 1] = $new_element
   $stack[0] += 1
EndFunc
```

Sklad je ustvarjen s svojimi standardnimi funkcijami *Create*, *IsEmpty*, *Peek*, *Pop*, *Push*. Funkcije *Peek* za ta primer ne bomo uporabili, vendar je vsebovana kot v vsakem popolnem skladu.

Funkcija *Stack_Create* naredi novo tabelo velikosti 5 mest. Prvo mesto v tabeli je rezervirano za indeks mesta, na katerem je zadnji dodani element v skladu. Sklad, ki ga dobimo s *Stack_Create*, ima torej velikost 4.

Funkcija *Stack_IsEmpty* preverja indeks zadnjega elementa. Če je indeks 0, to pomeni, da je sklad prazen.

Stack_Pop odstrani zadnji dodani element, zmanjša index zadnjega elementa za 1 in vrne element.

Stack_Push na tabeli uporablja AutoIt funkcijo *Ubound*, ki ugotovi njeno velikost. Velikost sklada se potem primerja z indeksom zadnjega elementa. Če sta enaka, potem je sklad izpolnjen do konca in nimamo prostih mest za nove elemente. V tem primeru povečamo tabelo za novih 5

mest brez izgube že shranjenih elementov (*ReDim*). Nov element se vstavi na prvo prosto mesto, indeks zadnjega elementa se poveča za 1.

Skripta ima dve glavni funkciji: *WinMinimize* in *WinUnminimize*. Prva skrči aktivno okno tako, da mu s funkcijo *WinSetState* spremeni stanje v skrčeno (@*SW_MINIMIZE*). Potem okno vrine na sklad. Kličemo jo z bližnjico <Ctrl> + <->. *WinUnminimize* sproži bližnjica <Ctrl> + <Shift> + <->. Če smo skrčeno okno medtem zaprli ali ga povečali na kakšen drugi način, se program ne sme ustaviti na tem oknu. Vrniti mora vedno naslednje okno, ki še vedno obstaja in je skrčeno. Zato se *WinUnminimize* sprehaja naprej po skladu in odstranjuje okna, dokler ne najde skrčenega oziroma pride do konca sklada. Če okno obstaja in je skrčeno, preveri stavek

```
BitAND(WinGetState($minimized_win_handle), 16)
```

WinGetState vrne stanje okna. Če okno obstaja in je skrčeno, je njegovo stanje označeno s številko 16. Okno ima lahko več stanj hkrati. *WinGetState* v tem primeru vrne binarni seštevek vseh stanj. Za to uporabljamo operator BitAND, ki preveri, ali trenutno stanje okna vsebuje stanje 16. Če je tako, potem je ustrezno okno najdeno. Funkcija *WinSetState* ga poveča in vrne nazaj na namizje (@*SW_RESTORE*).

7.3 Ročno premikanje in spreminjanje velikosti okna

Ta skripta nam bo dovolila premikanje in spreminjanje velikosti okna z miško, ko kliknemo katerikoli del okna. S tem se bomo znebili nerodnega klikanja na naslovno vrstico oziroma na tanke robove okna in dobili dve funkcionalnosti, ki ju že dolgo podpirajo upravljalniki oken za GNU/Linux. V Metacityju ročno premikanje in spreminjanje velikosti okna začnemo s klikom miškinega gumba, dokler držimo tipko <Alt>, kar pomeni da gumb sproži dejanje, če je tipka <Alt> aktivna. V tej skripti bo postopek obrnjen in bo bližnjica s tipkovnice sprožila dejanje, ko bo pritisnjen določeni miškin gumb. Z drugimi besedami, najprej bomo držali gumb miške in potem pritisnili bližnjico na tipkovnici. Razlog tiči v tem, da AutoIt še vedno ne podpira gumbov miške kot bližnjice, ki pokličejo neko funkcijo. Čeprav se je zaporedje aktivacij spremenilo, bomo poskrbeli, da bližnjica na tipkovnici ostane enaka in da gumbi miške določajo, katero dejanje se bo sprožilo. Levi gumb miške bo torej začenjal premikanje, srednji pa bo spreminjal velikost.

```
HotKeySet("!{space}", "StartAction")

While 1
   Sleep(900)  ;spi v ozadju in caka na klice bliznjic
WEnd

Func StartAction()
   If _IsPressed("001") Then
       WinDrag()
   ElseIf _IsPressed("004") Then
       WinResize()
   EndIf
EndFunc
```

Bližnjica <Alt> + <Space> pokliče funkcijo *StartAction*, ki naprej pokliče funkcijo *WinDrag* ali *WinResize*, odvisno od tega, ali držimo levi (`_IsPressed("001")`) ali srednji (`_IsPressed("004")`) gumb miške.

Prvi del skripte tvori funkcija *WinDrag*, ki skrbi za premikanje okna.

```
#include <Misc.au3>
#include <WinAPI.au3>

Opt("WinWaitDelay", -1)
Global $active_win_handle
Global $dll

Func WinDrag()
   $dll = DllOpen("user32.dll")
   $active_win_handle = WinGetHandle("[ACTIVE]")    ;poisce ID rocaj (oznako)
                                                    ;aktivnega okna
   Dim $win_start_position = WinGetPos($active_win_handle)
  ;vrne 4-dimenzionalno tabelo: {win_position_X, win_position_Y, win_width, win_height}
   Dim $mouse_start_position = MouseGetPos()
  ;vrne 2-dimenzionalno tabelo: {mouse_position_X, mouse_position_Y}
```

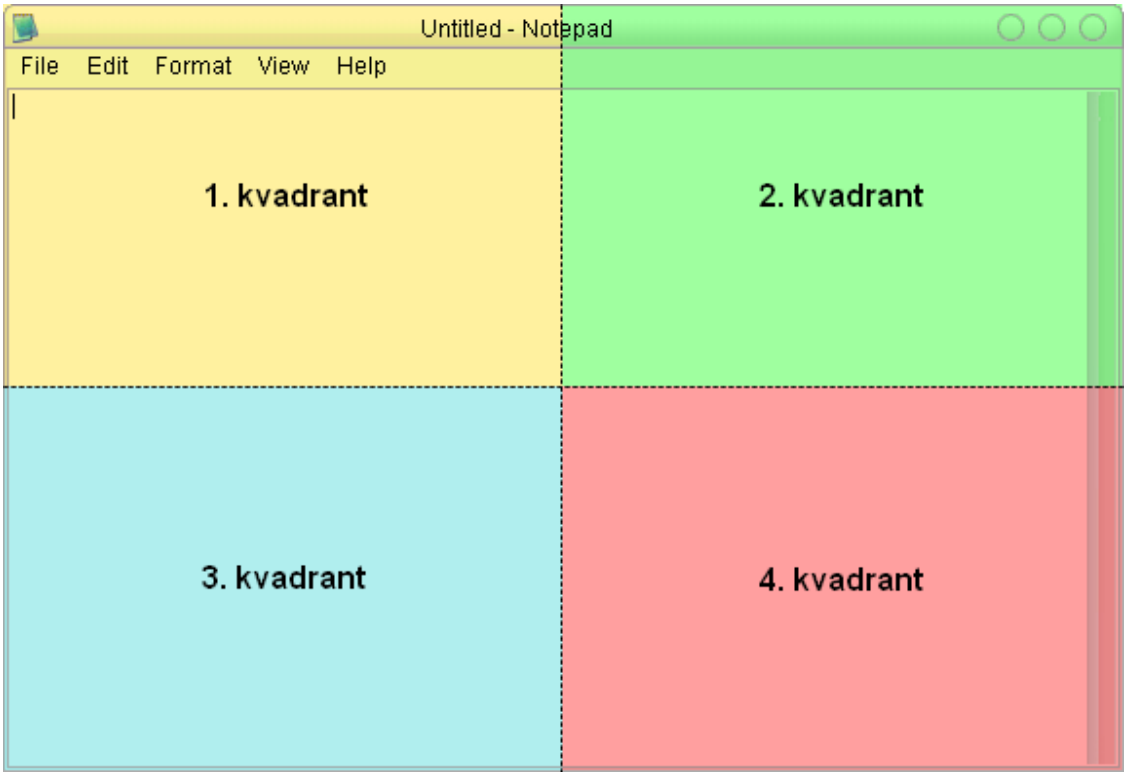
```
    While _IsPressed("001") ;dokler je levi gumb miske pritisnjen
        $mouse_position = MouseGetPos()
        $win_position = WinGetPos($active_win_handle)
        WinMove($active_win_handle, "", $win_position[0] + $mouse_position[0] -
              $mouse_start_position[0], $win_position[1] + $mouse_position[1] -
              $mouse_start_position[1])
        $mouse_start_position = $mouse_position
        Sleep (10)
    WEnd


    DllClose($dll)

EndFunc
```

WinGetPos vrne koordinate okna, *MouseGetPos* pa vrne koordinate kazalčka.

Glavni del funkcije *WinDrag* je zanka, ki teče, dokler držimo levi gumb miške. Znotraj zanke se po vrsti najprej ugotovi novi položaj kazalčka (*MouseGetPos*), potem trenutni položaj okna (*WinGetPos*) in potem se okno premakne na novi položaj (*WinMove*). Nov položaj se dobi z razliko med trenutnim položajem in razdaljo premika kazalčka. Iz uporabniške perspektive se nam zdi, kot da premikamo okno pod kazalčkom, v resnici pa premikamo le kazalček, ki mu sledi okno. Ko spustimo levi gumb miške, se zanka prekine, premikanje se ustavi.

V drugem delu skripte je funkcija za spreminjanje velikosti okna, *WinResize*. To je v bistvu razširjena funkcija *WinDrag*, ki razen položaja okna spreminja tudi njegove dimenzije na ustrezen način, ki je odvisen od tega, v katerem območju okna je naš kazalček. Okna bomo razdelili v štiri enaka območja oziroma kvadrante, kot to prikazuje slika 37. Na začetku preverimo, v katerem kvadrantu okna je kazalček in potem nadaljujemo postopek v zanki, podobni tisti iz funkcije *WinDrag*. Izvajati se začne eden od štirih različnih načinov spreminjanja velikosti, odvisno od kvadranta, v katerem smo. Med spreminjanjem velikosti tisti vogal, ki leži nasprotno od kvadranta, v katerem je kazalček, vedno ostaja negiben. Ker je položaj določen s prvo piko zgornjega levega vogala, bo postopek poleg spremembe dimenzij v nekaterih primerih vseboval tudi premikanje okna. Položaj okna bo treba spreminjati, če je kazalček v prvem

kvadrantu. Enako bo v primeru, da iz drugega kvadranta spreminjamo vertikalni delež velikosti ali iz tretjega horizontalnega. Samo v primeru, da dejanje začnemo znotraj četrtega kvadranta, se okno ne premika.



Slika 37: Območja za spreminjanje velikosti

Funkcija *WinResize* je sestavljena iz štirih delov kode, vsak pa vsebuje postopek za en kvadrant. V nadaljevanju bomo opisali le enega izmed njih, in sicer tistega, ki obravnava prvi kvadrant.

```
Opt("MouseCoordMode", 0)     ;koordinate pozicije kazalcka se iscejo relativno glede na
                             ;aktivno okno

If $mouse_start_position[0] < $win_start_position[2] / 2 And $mouse_start_position[1] <
$win_start_position[3] / 2 Then     ;1. kvadrant
   Opt("MouseCoordMode", 1)     ;koordinate pozicije kazalcka se obravnavajo
                                ;kot absolutne koordinate na zaslonu
   $mouse_start_position = MouseGetPos()
```

```
While _IsPressed("004") ;dokler je srednji gumb miske pritisnjen
    $mouse_position = MouseGetPos()
    $win_position = WinGetPos($active_win_handle)
    $declination_x = $mouse_position[0] - $mouse_start_position[0]
    $declination_y = $mouse_position[1] - $mouse_start_position[1]
    WinMove($active_win_handle, "", $win_position[0] + $declination_x,
          $win_position[1] + $declination_y, $win_position[2] - $declination_x,
          $win_position[3] - $declination_y)
    $mouse_start_position = $mouse_position
    Sleep (10)
WEnd
```

Na začetku nastavimo obravnavanje koordinat kazalčka relativno z aktivnim oknom (`Opt("MouseCoordMode", 0)`). To pomeni, da koordinate kazalčka ne bodo več oddaljenost od izhodišča namizja, temveč od izhodišča okna (prva zgornja leva pika okna). Nato pogoj preveri, ali smo res v prvem kvadrantu. To naredi tako, da preveri, ali je kazalčkova koordinata X manjša od polovice širine in ali je koordinata Y manjša od polovice višine okna. Če je to res, se postopek nadaljuje. Koordinate kazalčka spet nastavimo na absolutne (`Opt("MouseCoordMode", 0)`), ker bomo premikanju miške najprej sledili v celemu območju namizja. Sledi spreminjanje velikosti okna. Zanka deluje, dokler pritiskamo srednji gumb miške (`_IsPressed("004")`). Naprej poiščemo novi položaj kazalčka in sedanji položaj okna, nato izračunamo razdaljo, ki nam pove, za koliko se je kazalček od zadnjič premaknil. V funkciji *WinMove* zmanjšamo dimenzije z razdaljo, za to isto razdaljo pa premaknemo okno (zgornji levi vogal), tako da nasprotni kot prvega kvadranta pride spet v tisti položaj, v katerem je bil v predhodnem koraku zanke. Premikanje okna poskrbi, da nasprotni vogal nikoli ne spreminja položaja. Obenem da uporabniku občutek, da okno zmanjšuje ali povečuje iz zgornjega levega kota.

7.4 Pomikanje vsebine

Pomikanje vsebine okna (Angl. scrolling) je dejanje, ki ga danes namesto nerodne uporabe drsnih trakov na robu okna opravimo s pomočjo miškinega koleščka. Miškin kolešček je danes podprt v večini GUV-ov, vendar je z današnjimi velikimi količinami vsebin, ki jih dnevno pregledujemo,

še zlasti na medmrežju, postal prepočasna metoda. Njegov osnovni namen je pomikanje vsebine za samo nekaj vrstic gor ali dol. Edina preostala načina sta z omenjenimi drsnimi traki ali s tipkama <Page Up> in <Page Down>. Ker se želimo v duhu gest čim bolj oddaljiti od tipkovnice in se osredotočiti na miško, bomo s pomočjo AutoIta implementirali tri nove načine pomikanja vsebin. Vsi trije bodo simulirali vrtenje miškinega koleščka, a veliko hitreje in dlje. Premikanje bomo uredili s pomočjo gest, vendar ne tistih StrokeItovih. StrokeIt ne podpira tretjega tipa naprednih gest, tistega, s katerim lahko sprožimo in kontroliramo dejanje z enim in istim gumbom. Geste v StrokeItu služijo le za trenutna in avtomatična dejanja, kot je sprožanje ukazov, in niso del nobenega interaktivnega nadaljevanja. Zato bomo v času interaktivnega dejanja, kot je pomikanje vsebine, uporabljali geste, katerim bo sledila sama skripta.

7.4.1 Samodejno pomikanje

V tem načinu uporabnik nadzira le smer in hitrost, samo premikanje pa ni neposreden rezultat premikanja kazalčka. Z drugimi besedami, smer in hitrost sta relativni položaju kazalčka, spreminjanje položaja samega kazalčka pa ne opravlja pomikanja vsebine okna. Točka, v kateri začnemo pomikanje, je nevtralna in ko je kazalček v njej, se nič ne dogaja. Smer določamo, ko premaknemo kazalček višje oziroma nižje od začetne točke, hitrost pa je določena z njegovo oddaljenostjo od nje. Velja omeniti, da ta način pomikanja vsebine ni popolnoma nov. Najdemo ga v mnogih urejevalnikih in preglednikih dokumentov, spletnih brskalnikih itn. Redko je avtomatično pomikanje implementirano v vsem GUV-u. Takšno premikanje ni podprto recimo znotraj brskalnika datotek. Z AutoItom bomo rešili prav ta problem.

```
Global $active_win_handle
Global $mouse_start_position
Global $waiting = 0     ;pokaze, ce je pomikanje aktivno ali ne.
                        ; 0 = neaktivno, 1 = aktivno
Const $SENSITIVITY_1 = 25
Const $SENSITIVITY_2 = 90
Const $SENSITIVITY_3 = 260
Const $SPEED_1 = 120
Const $SPEED_2 = 40
Const $SPEED_3 = 10
```

```
Func AutoScroll()

    $active_win_handle = WinGetHandle("[ACTIVE]")    ;poisce ID rocaj (oznako)
                                                     ;aktivnega okna
    Dim $mouse_start_position = MouseGetPos()    ;vrne 2-dimenzionalno tabelo:
                                                 ;{mouse_position_X, mouse_position_Y}
    Local $scroll_declination_y = 0

    While _IsPressed("004")
        $mouse_position = MouseGetPos()
        $scroll_declination_y = $mouse_position[1] - $mouse_start_position[1]
        If $scroll_declination_y < -1 * $SENSITIVITY_3 Then
            MouseWheel("up", 1)
            Sleep($SPEED_3)
        ElseIf $scroll_declination_y > $SENSITIVITY_3 Then
            MouseWheel("down", 1)
            Sleep($SPEED_3)

        ElseIf $scroll_declination_y < -1 * $SENSITIVITY_2 Then
            MouseWheel("up", 1)
            Sleep($SPEED_2)
        ElseIf $scroll_declination_y > $SENSITIVITY_2 Then
            MouseWheel("down", 1)
            Sleep($SPEED_2)

        ElseIf $scroll_declination_y < -1 * $SENSITIVITY_1 Then
            MouseWheel("up", 1)
            Sleep($SPEED_1)
        ElseIf $scroll_declination_y > $SENSITIVITY_1 Then
            MouseWheel("down", 1)
            Sleep($SPEED_1)
        Else
        EndIf
    WEnd
EndFunc
```

Ko začnemo dejanje s pritisnjenim srednjim gumbom miške, gremo lahko s kazalčkom više ali niže od začetne nevtralne točke. Funkcija obravnava tri območja oddaljenosti kazalčka od te točke. Če smo v najbližji oddaljenosti, torej znotraj prvega območja, je pomikanje najpočasnejše. Ko izstopimo iz prvega in vstopimo v naslednje območje, je pomikanje hitrejše. Tretje območje je najhitrejše in ni navzgor omejeno. Parameter *SENSITIVITY_1* določa oddaljenost v pikslih od nevtralne točke, kje se začne prvo območje in s tem pomikanje. Meja med prvim in drugim območjem je določena s parametrom *SENSITIVITY_2*, *SENSITIVITY_3* pa določa mejo med drugim in tretjim območjem. Parametri *SPEED_1*, *SPEED_2* in *SPEED_3* so velikosti, podane v milisekundah, ki določajo hitrost pomikanja za posamezno območje. Večja vrednost pomeni počasnejše gibanje, saj ti parametri v bistvu določajo čas mirovanja med premikoma za eno vrstico gor ali dol.

Skripta je sestavljena iz treh parov podobnih delov kode. Vsak par pomeni eno območje, elementa znotraj para pa skrbita vsak za eno smer gibanja (gor ali dol). Na začetku vsake iteracije zanke (dokler držimo srednji gumb miške) se izračuna odklon kazalčka od nevtralne točke (*scroll_declination_y*). Potem se preveri, v katerem območju je kazalček. Lahko je više ali niže od nevtralne točke:

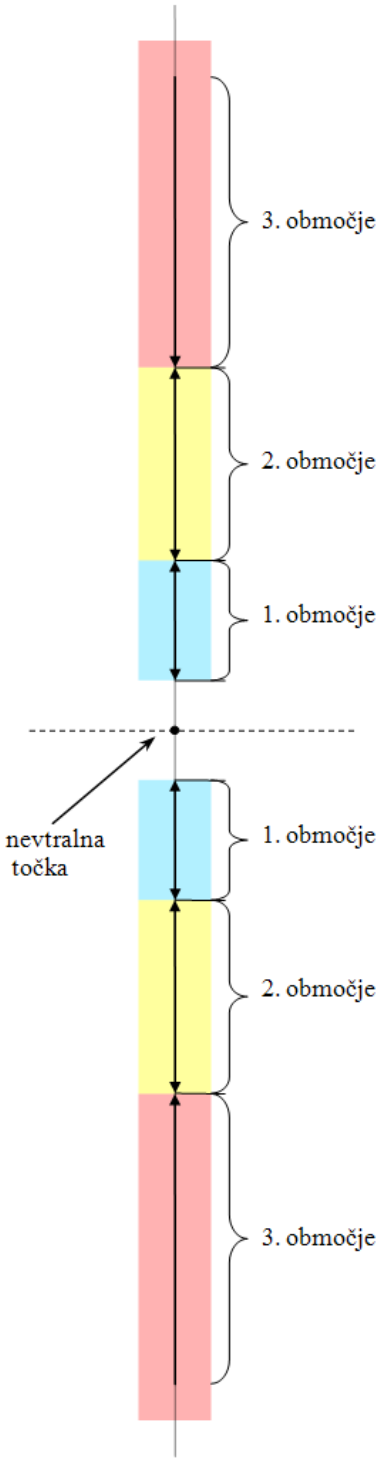
```
If $scroll_declination_y < -1 *
$SENSITIVITY_3
```

V skladu s tem simuliramo en premik miškinega koleščka v ustrezni smeri:

```
MouseWheel("up", 1)
```

Potem sledi mirovanje do naslednjega premika:

```
Sleep($SPEED_3)
```



Slika 38: Območja samodejnega pomikanja

7.4.2 Ročno pomikanje

Koda za ročno pomikanje bo simulirala miškin koleščk v času premikanja miške gor ali dol. Ta način bo v kombinaciji s StrokeItom deloval popolnoma enako kot dejanje za pomikanje v Easystroku, razen tega, da bomo v času dejanja držali srednji gumb miške namesto levega.

```
Func ManualScroll()
   Dim $mouse_position
   While _IsPressed("004") ;dokler je levi gumb miske pritisnjen
       $mouse_position = MouseGetPos()
       If ($mouse_position[1] - $mouse_start_position[1]) > 0 Then
           MouseWheel("down", $MANUAL_SPEED)
       ElseIf ($mouse_position[1] - $mouse_start_position[1]) < 0 Then
           MouseWheel("up", $MANUAL_SPEED)
       EndIf
       $mouse_start_position = $mouse_position
       Sleep (10)
   WEnd
EndFunc
```

Dokler držimo srednji gumb, zanka preverja spremembo koordinate Y v položaju kazalčka. Če je sprememba od zadnjega položaja pozitivna (torej se je miška premaknila navzdol), program simulira vrtenje miškinega koleščka dol in obratno. "Hitrost" pomikanja oziroma število premikov koleščka ob vsakem koraku zanke je določeno s parametrom *MANUAL_SPEED*.

7.4.3 Navijajoče pomikanje

Ročno pomikanje nam omogoča večji nadzor od avtomatičnega, vendar hitro ostanemo brez prostora in pogosto zadenemo zgornji ali spodnji rob namizja. Zato je naslednji koncept lahko najuporabnejši med vsemi tremi. To je pomikanje, ki je podobno ročnemu, vendar nikoli ne ostanemo brez prostora, ker miško vrtimo v krogu. Poimenovali ga bomo "navijajoče", ker z vrtenjem miške v smeri urinega kazalca oziroma nasproti lahko dobimo občutek, da navijamo nekakšen škripec, ki vleče vsebino okna gor ali dol. Smer urinega kazalca bo za sprehajanje navzdol po vsebini, nasprotna smer pa za navzgor.

```
Func WindingScroll()
    $mouse_start_position = MouseGetPos()
    Dim $mouse_position
    Dim $r1 = 0, $r2 = 0
    While _IsPressed("004") ;dok je levi gumb miske pritisnjen
        $mouse_position = MouseGetPos()

        If ($mouse_position[0] - $mouse_start_position[0]) > 0 Then
            $r2 += 10
        EndIf
        If ($mouse_position[1] - $mouse_start_position[1]) > 0 Then
            $r2 += 1
        EndIf

        If $r1 = 10 And $r2 = 11 Then
            MouseWheel("down", $WIND_SPEED)
        ElseIf $r1 = 10 And $r2 = 0 Then
            MouseWheel("up", $WIND_SPEED)
        ElseIf $r1 = 11 And $r2 = 1 Then
            MouseWheel("down", $WIND_SPEED)
        ElseIf $r1 = 11 And $r2 = 10 Then
            MouseWheel("up", $WIND_SPEED)
        ElseIf $r1 = 1 And $r2 = 0 Then
            MouseWheel("down", $WIND_SPEED)
        ElseIf $r1 = 1 And $r2 = 11 Then
            MouseWheel("up", $WIND_SPEED)
        ElseIf $r1 = 0 And $r2 = 10 Then
            MouseWheel("down", $WIND_SPEED)
        ElseIf $r1 = 0 And $r2 = 1 Then
            MouseWheel("up", $WIND_SPEED)
        Else
        EndIf

        $mouse_start_position = $mouse_position
        $r1 = $r2
        $r2 = 0
```

```
        Sleep (10)
    WEnd
EndFunc
```

Koda obravnava krožnico gibanja kazalčka v štirih delih. Glede na sosledje delov je določeno pomikanje navzdol oziroma navzgor.

7.4.4 Več funkcionalnosti znotraj ene skripte

Do zdaj smo obravnavali posamezne skripte, kjer je bila za vsako posebej določena bližnjica, ki kliče glavno funkcijo. Seveda je mogoče vse skripte povezati v en sam program, ki sprejema več bližnjic, in na ta način omogoča več različnih funkcionalnosti hkrati. Ena prednost pri tem je, da lahko poganjamo le en program z razširitvami našega GUV-a in na tak način varčujemo pomnilniške vire. V nadaljevanju bomo pokazali del kode, ki nastavlja več funkcij pod eno bližnjico. Ko uporabimo bližnjico, program čaka na vhod iz miške. S pomočjo gumbov miške bomo pred začetek pomikanja vsebine dodali možnost izbire med tremi načini.

```
Func CheckScrollStatus()
    If ($waiting = 0) Then  ;ce ze ni bilo sprozeno pomikanje
        $waiting = 1
        While 1
            ToolTip("Nacin pomikanja")
            If _IsPressed("004") Or _IsPressed("001") Then
                $mouse_start_position = MouseGetPos()
                ;ToolTip("")
                ExitLoop
            EndIf
            Sleep(10)
        WEnd
```

... Prva zanka za izbiro: Samodejno pomikanje ...

```
        If Not _IsPressed("004") Then
            $waiting = 0
        EndIf
        ToolTip("")
    EndIf
EndFunc
```

Funkcija *CheckScrollStatus* bo tista, ki jo kliče bližnjica. Najprej se preveri, ali je že bila poklicana in je ena njena instanca že v stanju čakanja; za to nam služi spremenljivka *waiting*. Ko je funkcija sprožena, se *waiting* nastavi na 1. Sledi zanka, ki čaka na vhod z miške. Majhno pojavno obvestilo, ki se prikaže ob kazalčku (*ToolTip*) nas opozori, da glavno dejanje poteka in program čaka na vhod za nadaljevanje. Ko pritisnemo srednji ali levi gumb miške, se poišče položaj kazalčka (*mouse_start_position*) in zanka se prekine, kar omogoči nadaljevanje programa. Sledijo zanke za pomikanje. Poleg srednjega gumba, ki omogoča nadaljevanje in je pomemben v zankah kasneje, levi gumb služi samo kot dodatna možnost prekinitve programa. Prva zanka za izbiro pomikanja se požene, če držimo srednji gumb. Če smo ga spustili ali pritisnili levega, se pomikanje ne požene in funkcija *CheckScrollStatus* konča svoje delo, a predhodno še nastavi waiting na ničlo, kar jo odklene za vnovično uporabo.

```
While _IsPressed("004") ;1. zanka za izbiro: samodejno pomikanje
    ToolTip("SAMODEJNO pomikanje")
    $mouse_position = MouseGetPos()
    If $mouse_position[1] <> $mouse_start_position[1] Then  ;ce se je miska
                                                            ;zacela premikati
        ToolTip("")
        AutoScroll()
        ExitLoop
    ElseIf _IsPressed("001") Then   ;ce pritisnemo levi gumb prvic
        Do
            Sleep(10)
        Until Not _IsPressed("001") ;mirovanje dokler ne spustimo levega
                                    ;gumba
```

... Druga zanka za izbiro: Ročno pomikanje ...

```
   EndIf
      Sleep(10)
Wend
```

Če smo v predhodnem koraku pritisnili in držali srednji gumb miške, je program nadaljeval in pognal prvo zanko za izbiro načina pomikanja. Pojavno obvestilo ob kazalčku se spremeni in nas obvesti, da smo trenutno v samodejnem načinu. V vsaki iteraciji zanke se preverja, če smo premaknili miško ali pritisnili levi gumb. Če miško premaknemo, program sklepa, da smo začeli pomikanje, požene funkcijo za samodejni način (*AutoScroll*), zanka se prekine in funkcija *CheckScrollStatus* se konča. Če smo brez premikanja miške kliknili levi gumb, gre program v drugo fazo. Najprej počaka, da dokončamo levi klik (spustimo levi gumb), nato nadaljuje z zanko za izbiro drugega načina. Pomembno je, da čez ta prehod držimo srednji gumb.

```
While _IsPressed("004") ;2. zanka za izbiro: rocno pomikanje
   ToolTip("ROCNO pomikanje")
   $mouse_position = MouseGetPos()
   If $mouse_position[1] <> $mouse_start_position[1] Then  ;ce se je miska
                                                           ;zacela premikati
       ToolTip("")
       ManualScroll()
       ExitLoop
   ElseIf _IsPressed("001") Then   ;ce pritisnemo levi gumb drugic
       Do
           Sleep(10)
       Until Not _IsPressed("001") ;mirovanje dokler ne odpustimo levega
                                   ;gumba
```

... Tretja zanka za izbiro: Navijajoče premikanje ...

```
       ExitLoop    ;izhod iz druge zanke
   EndIf
       Sleep(10)
Wend
```

Druga zanka deluje enako kot prva. Pojavno obvestilo nam pove, da smo v ročnem načinu. Če začnemo premikati miško, se zažene funkcija ročnega načina (*ManualScroll*), *CheckScrollStatus* pa se konča. Če namesto tega zdaj drugič kliknemo levi gumb, program nadaljuje s tretjo izbiro.

```
While _IsPressed("004") ;3. zanka za izbiro: navijajoce pomikanje
   ToolTip("NAVIJAJOCE pomikanje")
   $mouse_position = MouseGetPos()
   If $mouse_position[1] <> $mouse_start_position[1] Then  ;ce se miska zacela
                                                           ;premikati
       ToolTip("")
       WindingScroll()
       ExitLoop
   ElseIf _IsPressed("001") Then   ;ce pritisnemo levi gumb tretjic
       Do
           Sleep(10)
       Until Not _IsPressed("001") ;mirovanje dokler ne odpustimo levega
                                   ;gumba
       ExitLoop    ;izhod iz tretje zanke
   EndIf
Wend
```

Zanka za navijajoči način deluje podobno, s to razliko, da nas ne pelje dalje v globino. Če takrat tretjič kliknemo levi gumb, nas ta vrže iz tretje zanke nazaj v prvo. Ker prva nima stavka *ExitLoop*, se potem spet najdemo v prvi izbiri oziroma samodejnem načinu, seveda ob predpostavki, da ves čas držimo srednji gumb miške.

8 SKLEP

Vsak dan tehnologija postaja vse bolj napredna, močna in vsestranska. Zasuti smo z ogromno količino informacij, ki vedno naraščajo in jih človek vse težje sprejema. Zato je HCI posebej pomembno področje, ki ga je treba razvijati na način, da navaden človek problem lažje obvladuje. Da bi omogočili še hitrejše in časovno učinkovito delo naprednim uporabnikom, dobro privajenim na grafični vmesnik, ter povečali hitrost učenja in privajanja začetnikov na uporabo računalnika, moramo vedno izboljševati vmesnike. To je treba delati na način, da nove razširitve in posodobitve vedno povečajo enostavnost in intuitivnost ter predvsem zmanjšajo količino kognitivne obremenitve uporabnika med opravljanjem posameznih nalog.

V 30 let razvoja grafičnih uporabniških vmesnikov se je ohranjala udomačenost dizajna namesto da bi uvajali dobre konceptualne spremembe, ki bi izboljšale intuitivnost in pripeljale vmesnike na novo raven. Čeprav so HCI eksperti, kot sta W. Buxton in J. Raskin, opozarjali na potrebne spremembe in predlagali svoje rešitve, do radikalnih sprememb v razvoju popularnih GUV-ov nikoli ni prišlo. V nadaljevanju njihove zapuščine in v duhu odpora proti zastoju v dizajnu GUV-a so v tem delu predstavljene razširitve popolnoma sprejemljive glede na ceno in enostavnost uporabe. Te upravljalne razširitve GUV-ov operacijskih sistemov GNU/Linux in Windows torej upoštevajo pogoje dostopnosti, nizke cene, intuitivnosti, enostavnosti uporabe in pospešitve dela na računalniku. Kot glavna sprememba v dosedanjih GUV-ih so uporabljene geste z miško. Geste na splošno so človeku prirojen obrazec komunikacije, ki dokazano zmanjšuje obremenitev delovnega spomina, omogočajoč umu zbranost za osnovne probleme. Kot take so maksimalno intuitivne in idealne za kognitivno razbremenitev računalniških uporabnikov.

Bistvo predstavljenih gest z miško je podati neizkušenem uporabniku čim večji in čim enostavnejši nadzor nad računalnikom. Za naprednejše uporabnike je cilj znebiti se počasnega iskanja in nerodnega ciljanja majhnih upravljalnih gradnikov GUV-a. Namesto iskanja nadzornih podob po zaslonu imamo lahko ves nadzor na dlani. Počasno vizualno orientacijo zamenja kognitivna avtomatizacija.

Eden izmed ciljev diplomske naloge je pokazati, da je s pomočjo teh razširitev možno ustvariti edinstven sistem upravljanja na različnih operacijskih platformah. S tem geste odkrivajo

potencial premagovanja ovir, ki uporabniku na prehodu med platformami zastavljajo njihove implementacije različnih grafičnih vmesnikov.

Začetni del predstavitve razširitev GNU/Linux GUV-a obravnava upravljalnik oken Metacity. Funkcionalnosti Metacityja, ki so sicer zelo koristne, ampak bolj skrite pred uporabnikom, so prikazane s konfiguracijo. Obravnava se predvsem upravljanje z okni. Uporabljene funkcionalnosti so povezane z določenimi bližnjicami, ki so razporejene na tipkovnici tako, da približno ustvarjajo koncept akordične tipkovnice, še enega pripomočka za hitro in poenostavljeno upravljanje, ki ga je poleg miške prvič predstavil Douglas Englebart. Funkcionalnosti so v naslednjem delu pripeljane na višjo raven uporabe s pomočjo gest z miško. Akordična razporeditev bližnjic je obdržana in tako je ohranjena možnost njihove uporabe z eno roko: tisto, ki je na miški. Drugi roki je podarjena moč gest za upravljanje sistema, ki jih omogoča brezplačni program Easystroke. Namestitev gest je ponazorjena skozi proces konfiguriranja Easystroka, s katerim so predstavljene možnosti programa. Easystroke podpira več tipov ukazov, ki se lahko sprožijo ob uporabi gest. Vsak tip je ustrezno predstavljen. Opisani in prikazani sta dve vrsti gest: enostavne in napredne. Geste obeh vrst so osmišljene na način, ki ohranja čim večjo enostavnost, intuitivnost in medsebojno raznovrstnost. Ta načela zagotovijo hitrost uporabe, lažji in razbremenjen spomin ter zanesljivost in nedvoumnost pri uporabi.

V nadaljevanju so predstavljene razširitve GUV-a operacijskega sistema Windows. StrokeIt, brezplačni program za geste v Windowsih, je razložen skozi konfiguracijo. Geste in ustrezna dejanja so pazljivo izbrana tako, da čim bolj zvesto preslikajo konfiguracijo iz GNU/Linuxa. Na tak način je dobljeni model upravljanja z gestami ekvivalenten tistemu z druge platforme. V pomanjkanju nekaterih možnostih pri StrokeItu, ki jih ima Easystroke, je uporabljen skriptni jezik AutoIt. Manjkajoče funkcionalnosti so z uporabo skript dodane operacijskemu sistemu brez direktne odvisnosti o StrokeItu. To na koncu diplomskega dela ponazarja, kako lahko s svojimi rokami sami razširimo in obogatimo GUV operacijskega sistema.

V trenutku, ko si nove in precej drage tehnologije vizualnega sledenja gibom telesa (kot je Microsoft Kinect) in številne implementacije zaslonov na dotik prizadevajo zagospodariti nad prihodnostjo človeško-računalniške interakcije, to diplomsko delo prinaša enostavne in poceni rešitve za posodobitev sistema človeškega upravljanja osebnega računalnika. Tisti, ki jih uporabljamo vsak dan, poznamo njihovo resnično moč. Če se vendarle na koncu zgodi, da vmesniki, občutljivi na gibanje, zares postanejo prihodnost, smo prav mi tisti, ki jo bomo prvi obvladali.

9 VIRI

[1] J. Reimer, *A History of the GUI*, arstechnica.com, May, 2005.

[2] O modelu WIMP. Dostopno na: http://en.wikipedia.org/wiki/WIMP_(computing)

[3] W. Buxton, "Chunking and Phrasing and the Design of Human-Computer Dialogues", v zborniku *Proceedings of the IFIP World Computer Congress*, Dublin, Ireland, 1986, str. 475-480.

[4] J. Reimer, *A History of the GUI*, arstechnica.com, May, 2005.

[5] O znanosti gest. Dostopno na: http://www.parentingscience.com/gestures.html

[6] J. Raskin, *Intuitive equals familiar*, Communications of the ACM. 37:9, September 1994, str. 17.

[7] Tržni deleži operacijskih sistemov. Dostopno na: http://statowl.com/operating_system_market_share.php

[8] O namiznem okolju GNOME. Dostopno na: http://en.wikipedia.org/wiki/GNOME

[9] O priljubljenosti operacijskih sistemov. Dostopno na: http://www.seopher.com/articles/popularity_graphs_of_the_entire_operating_system_market_ubuntu_vista_os_x_and_more_

[10] Tipi GDK dogodkov. Dostopno na: http://www.gtk.org/tutorial1.2/gtk_tut-29.html

[11] Splošno o orodju StrokeIt. Dostopno na: http://www.tcbmi.com/strokeit/

[12] Seznam privzetih gest orodja StrokeIt. Dostopno na: http://www.tcbmi.com/strokeit/wiki/index.php?title=Help:Gesture_list

[13] Domača stran AutoIt skriptnega jezika. Dostopno na: http://www.autoitscript.com/site/autoit/