Classic AmigaOS 4 MorphOS AROS UAE

AMIGA FORUM

www.suga.se
www.amigaforum.se
DEC. 2014 #011

STOR NYBÖRJARGUIDE I HOLLYWOOD

• Nyheter • SAFIR 10 år • SUGA •
• Generation 64 • Generation Amiga? •

Jag vill börja denna ledare med att be om ursäkt att detta nummer blivit försenat. Det beror på en kombination av tidsbrist, ett överraskande svårt översättningsarbete samt julledigt. Jag hoppas ni har överseende med detta.

Så länge jag kan minnas har jag velat kunna programmera. Och lite då och då har jag gjort seriösa försöka att lära mig, men alltid fastnat så gott som omedelbart. Nybörjarböcker råder det ingen brist på, men av någon anledning så börjar de nästan alltid enkelt för att sedan helt plötsligt utgå från att nybörjaren förstår hur en dator "tänker" och hoppar till mer avancerade saker. Och där sitter jag och kliar mig på huvudet och känner mig bara dum.

Jag ville råda bot på det och bad en Hollywood-programmerare att skriva en nybörjarguide för folk helt utan programmeringserfarenhet och han skrev en lättbegriplig och jättelång sådan. Efter att ha läst den och testad alla exemplen förstår jag mycket bättre hur grundläggande programmering fungerar och jag känner mig sugen på att göra något eget. Förhoppningsvis kommer du läsa den och tycka likadant. Vår pyttelilla datorvärld behöver fler programmerare, och program.

Johannes Genberg

Moturs från längst fram: zIMzUM, z-nexx, CD32, Filibus, Bearsoft, Mr Bad, Bittin, Wedin, SMF och MagnusG på SAFIR:s 10-årsjubileum (foto: Stefan Nordlander)

Nummer 11 2014

Omslag: Smoker av Fairfax (64 färger, beskuren, 1994)

Amiga Forum ges ut av föreningen Swedish User Group of Amiga (SUGA). Tidningen utkommer 4 ggr/år. Artiklar och föreningsinformation skickas till redax@amigaforum.se. Deadline är den 15 februari, maj, augusti och november. Lösnummer säljs via suga.se/amigaforum/skaffa.html. Maila redaktionen om din amigaförening också vill prenumerera.

Rapport från Safir (safir.amigaos.se)

Vårt kära förbund har nu passerat tio-års strecket. Och visst kan man bara hålla med om att vi har mognat en hel del längs vägen! Både som individer, men framförallt som en helhet – som ett community. De av oss som varit med från början vet hur hetsiga diskussionerna kunde vara, hur spydiga och omogna kommentarer kunde svida och hur sura miner kunde hålla i sig i dagar om inte veckor. Idag har vi en betydligt lugnare och harmoniskare stämning, där vi även om det brusar upp ibland oftast löser konflikterna relativt enkelt, oftast helt utan moderatorns ingripande.

Den 14/11 firade vi på dagen Safirs 10-års jubileum i Syntax Societies fina lokaler i Landskrona. Det blev en sprakande tillställning (nåväl, så sprakande det kan bli när femton Amiganördar samlas i en källare för att spela dataspel och snacka skit). Det bjöds på burgare från grillen och dagen till ära hade vi fixat en Safir-tårta som besökarna fick avnjuta.

Flertalat stannade kvar över helgen och på lördagen arrangerades även två föreläsningar i FPGA/VHDL utveckling samt Commodore C=64 demoscenehistoria. Det var ett mycket lyckat event och vi ser så klart fram mot många kommande fantastiska år med er på *http://safir.amigaos.se*!

Safirs Moderatorer!

Rapport från SUGA (suga.se)

Lika varm som sommaren var så har hösten varit mörk. Detta är något som vår fönsterlösa lokal lämpar sig utmärkt för. Som en uppföljning från förra nummret kan jag berätta att vi nu har tagit över hela lokalen. Vi har också övertagit en del skrot som tidigare ockupant lämnat efter sig, men så snart någon med bil med dragkrok får tid så ska dessa saker bort. Det blir även lite möbelbygge. Ett passande arbetsbord och, om vi inte hittar något lämpligt färdigbyggt, en platsbyggd lagerhylla.

Med den andra lokalhalvan fick vi även ett kallförråd. Det var nog inte ordentligt städat på 100 år, men nu kan man ha saker där utan att dom blir skitiga (Amigorna håller vi såklart fortfarande i den isolerade delen av lokalen). Till mitten av lokalen har vi fått ett nytt stabilt bord och till soffan en tillhörande TV. Det är en 28" tjock-TV som är kopplad till en A1200 med SCART, utmärkt för tvåspelarspel.

Lokalens senaste tillskott är en internetuppkoppling. Det är inte riktigt internet, för det går ingen kabel mellan våra datorer och resten av internet. Hur dessa Voodoo-konster går till är oklart, men på något sätt kan Mies (oftast) komma åt internet nu och har därför konfigurerats till att dela ut detta vidare till våra riktiga datorer, med riktiga datachips, fasta IP:n och TP och BNC.

Större lokal betyder fler uppställda Amigor och eftersom vi gillar nätverk har vi skaffat ett nytt nätverkskort, ett X-Surf 100 från GGS-Data. Vår A3000 Ruri visade sig dock bara ha Buster 7, så en revision 11 har införskaffats.

På aktivitetsfronten har det inte hänt något spektakulärt. Saker och ting har rullat på som vanligt och det är inget fel med det, vi har ju trevligt. Kjell har fixat med och svurit över PC-emulatorer, Iggy har ätit nudlar och programmerat, undertecknad har lagat och sålt en A2000 och ett löst moderkort och vi har alla hjälpts åt att sortera vår stora samling Amigatidningar. Fler Amigor har lagats, PC-diskettenheter konverterats till Amiga, Superted lidits av och KVM-switchen har fixats så att man kan använda den utan PS/2-tangentbord.

Kort sagt, föreningen är ganska välmående och aktiviteten fortsätter varje lördag. Det är lite rörigt under tiden som vi flyttar in i den nya lokalhalvan, men låt inte det hindra om du vill titta förbi och köra lite Amiga. God Jul!

Jonas Hofmann

En 10-årig ädelsten

Sveriges största Amigaportal fyller ett decenium

Av Stefan Nordlander

Med risk för att låta dramatisk - kära vänner, kan ni tänka er! Det har nu gått *tio år* sedan vi startade Safir! Helt fantastiskt. Det känns som igår jag och Andreas 'reflect' Loong satt där i MA265 bland hyllor och kartonger i Mölndal utanför Göteborg och smidde planer. Många av er har säkert hört den historien så det blöder ur öronen på er, men för er som inte känner till hur det startade; och *varför* det startade så kan jag avslöja att det för tio år sedan inte alls var guld och gröna skogar i Amigaland.

Nej, AmigaOS.SE registrerades som en tydlig markering att detta, *Detta* var minsann en portal av och för oss som körde det "äkta" AmigaOS. Inga "kopior" eller x86-spinoffs. Det är ingen idé att borsta det faktumet under mattan idag. Idag ser jag personligen domänen som ett viktigt arv för att visa var vi kommer från. Vad vi alla har gemensamt, oavsett val av hård eller mjuk plattform. Vi kommer alla från AmigaOS-plattformen oavsett om vi idag kör AmigaOS4, AROS eller MorphOS.

Hagbard Handfaste (© Dik Browne 1973) var inspirationsskällan till Safirs egna viking (se förra sidan).

Men det var givetvis ingen anledning i sig att starta en portal och ett forum! Nej, det hade länge talats om ett gemensamt "förbund" för svenska Amiga-föreningar och dess like (tydligen) på den tidens populäraste Svenska Amigaforum www.amigarulez.se. Enda problemet med dessa diskussioner var att man aldrig blev överens! Tiden gick och tjafset bara fortsatte... "Lägren" bråkade om allt från namnet på forumet till färgen på T-shirtarna och vad man skulle ha för pålägg på mackan på morgonen.

Vi som stod vid sidan av och var åskådare till detta spektakel bara skakade på huvudet. Vi var några som oftare hängde på de utländska forumen så som www.amigaworld.net, men sanningen är att det var lika illa om inte värre där. AmigaWorld gick så långt att moderatorerna delades upp vid ett tillfälle och några gick och startade www.amigans.net som då var ett "renodlat" AmigaOS-forum, utan kategorier för "alternativa plattformar". Och så är det väl än idag om jag inte förstått det hela fel?

Hur som helst var det i denna veva som jag en dag efter att ha läst någon tråd på Amigarulez bara fick nog och gick in på iis.se och registrerade amigaos.se, kontaktade några representanter

i communityt och frågade om de ville vara med på tåget. En mycket viktig spelare på planen här är också saebla så klart! Historien här är att han i samband med allt detta skulle göra sitt examensarbete för Mittuniversitetet i Sundsvall. Vi kände varandra redan tidigare dels för att min bror gick i samma klass, samt att han var aktiv i dataföreningnen MHD som jag var styrelsemedlem i under min studietid på MIUN.

Vi kom raskt överens om att han skulle bygga en portal baserad på L.A.M.P. (Linux, Apache, MySQL & PHP) till oss och projektnamnet skulle vara SAFIR; *Svenska AmigaFöreningar I Riket*.

Ganska länge snurrade hela forat på en liten Dell Optiplex under mitt skrivbord på min 8/2 Mbit ADSL-lina. På den tiden var vi väl hundra medlemmar eller så. Men efter min lina dippat för tjugo-elfte gången och en fläkt stannat i min PSU (och smält!!!) så forat gick ner över en helg var det dags att flytta till lite proffsigare grejer. Det fick då bli en fantastisk 2xPIII Compaq-server med 512MB RAM som monterades *vertikalt* på väggen i Syntax Societys serverrum. Där sitter den än i dag faktiskt. Här rullade Safir i ett par år tills vi tack vare en medlem som då jobbade på FSData fixade ett tids-obegränsat DeLux-Webhosting-avtal till oss för exakt 0 kr. Vi är alltså obegränsat sponsrade av FSData! Därav bannern på förstasidan.

Namnet ”Svenska Amigaföreningar i Riket” var det flera personer inblandade i om jag inte minns fel. Men ”the final touch” var det StudentK & Jonas som hade, när de kom på den fina förkortningen S.A.F.I.R. (Rätta mig om jag har fel här). Så på den vägen är det. Vi jobbade ganska länge i projektform med att sätta upp en liten organisation för SAFIR. Vi hade regelbundna möten på IRC, och förde protokoll, och så vidare Med fanns representanter för varje förening som var med: ACG, ACGGbg, SUGA och Syntax Society var väl de som var med först tror jag. Senare gick ACG-G och GCO också med i förbundet. Men sanningen är att idag lever SAFIR mer sitt eget liv än via sina respektive föreningar.

Några korta ord om vår maskot – Olaf. Den första versionen som kom till för tio år sedan. Jag har svaga minnen av hur vi valde den. Men det var i alla fall Kjell ’Sharakmir’ Bredig som ritade denna. Den är så klart ogenerat baserad på *Hagbard* som ägs av Bull Press. Vi har kollat med dem huruvida det är OK för oss att använda vår Olaf här på forat, och har fått godkänt för detta. Men däremot tyckte de inte att vi skulle trycka T-shirts och sälja då han var lite ”för lik” deras Hagbard på tunnan. Hurvida detta skulle hålla i rätten eller ej låter jag någon annan tvista om, men onekligen har vi ju använt Hagbard som inspiration. Det var mycket därför vi nu i år, lagom till 10-årsjubileumet valde att anlita Liksmaskaren här på forumet för att ge Olaf ett helt nytt fräscht utseende!

Idag är sanningen den att ”Safir” – som vi väljer att skriva det (som ett namn istället för en förkortning) – går ganska mycket på sparlåga som ett förbund. Men det är *ni* – *våra medlemmar* – som håller portalen vid liv. Och det är precis så som det skall vara. Precis som vi vill ha det! Visst, vi skriver lite nyheter ibland när andan faller på, och vi moderatorer finns här i bakgrunden när vi behövs och vi läser och skummar igenom det mesta (men långt från allt, det hinner vi inte). Skulle någon tvist uppstå så finns vi här för att hjälpa till att lösa den. Men i det stora hela lever Safir sitt eget liv, och ni sköter det riktigt bra!

Så från oss till er; *tack* för att ni göra Safir till ett sådant *fantastiskt* forum att vistas i!!

Nyheter från Amiga-världen

AmigaOS 4: Ny version av AmigaOS här

Efter 6 års gratis uppdateringar kommer nu en ny version av AmigaOS 4.1 – Final Edition (FE). Förutom att den innehåller alla uppdateringar till dags datum samt den nya Ubooten för SAM 460-moderkortet innehåller den också flera nya funktioner som ursprungligen var planerat för 4.2.

Den kostar 30 euro och kräver inte att man äger en tidigare version av AmigaOS 4.1. FE har börjat säljas från och med december 2014.
https://amigaos.net

MorphOS: Stöd för SAM 460 på väg

MorphOS Team brukar inte avslöja vad de jobbar med, men det är bekräftat att de håller på att porta operativsystemet till SAM 460-moderkortet.

Anledningen antas vara den utbredda kritiken att man inte kan köpa nytillverkad hårdvara till MorphOS, vilket nu alltså åtgärdas.

Något releasedatum är inte satt.
http://morphos.de

Business: A-EON och AmigaKit köper in sig

Efter köpet av Amiga.org, Amigaworld.net och Personal Paint har A-Eon och AmigaKit nu även köpt 3D-renderingsprogtrammet Aladdin 4D i syfte att vidareutveckla det.
http://a-eon.biz/?news=12-12-2014

NG: Nytt 3D-spel på väg

Cherry Darling har avslöjat exklusivt för Amiga Forum att de jobbar på ett nytt 3D-spel för alla NG-Amigor. Det kommer heta *Wings Battlefield* och är en FPS-flygsimulator med första världskrigs-tema.

Den stora nyheten är att den kommer ha hårdvaru-3D-stöd för AmigaOS 4 med RadeonHD-kort, tack vare egenutvecklade drivrutiner för spelet. Har man inget sånt kort stöds Warp3D istället.
http://cherry-darling.net

Classic: Nya Workbench 3.1-disketter

Workbench-disketter har varit slutsålda länge. Därför har AmigaKit, i samarbete med Cloanto, nu släppt en ny uppsättning floppydisketter.

Dessa disketter innehåller också smärre förbättringar som Y2K-buggfix och fullt stöd för A4000T.
http://amigakit.leamancomputing.com

Classic & AmigaOS 4: Personal Paint 7.3 släppt

En ny version av PPaint är släppt. Förutom nya finesser så är den största nyheten native-stöd för AmigaOS 4.1.

Den kostar drygt 20 brittiska pund. Man kan också ladda ner 7.1 gratis från hemsidan nedan.
https://www.ppaint.com

UAE: WinUAE 3.0 med PPC-stöd släppt

Förutom mängder med buggfixar och förbättringar är den största nyheten stöd för PPC-kort. Detta möjliggör att man kan köra AmigaOS 4.1 Classic i emulerat läge, vilket varit efterlängtat.
https://www.winuae.net

Classic: BetterWB 3.9 släppt

BetterWB är en samling patchar och uppdateringar för ett nyinstallerat Workbench 3.1. Den innehåller inget ögongodis utan alla förbättringar ligger under skalet. Version 3.9 är ute nu och har väldigt blygsamma hårdvarukrav.
http://lilliput.amiga-projects.net/BetterWB.htm

MorphOS: Ny ordbehandlare på väg

Carsten Siegner, som utvecklade den ofärdiga ordbehandlaren *Scriba*, har börjat med en ny ordbehandlare som nu har kommit ut i version 0.4.
http://world-of-amiga.eu

AmigaOS 4: Appstore öppnad

Nu har även AmigaOS 4-ägare fått en appstore. Via den kan du bland annat köpa kommersiell mjukvara. Du måste ladda ner ett program för att kunna använda affären. Notera att det fortfarande är i beta-stadiet.
https://www.amistore.net

AmigaOS 4: RadeonHD 2.4 på väg

A-Eon har anlitat Hans de Ruiter för att uppdatera Radeon-drivrutinerna. Denna version kommer bland annat ha stöd för R5, R7 och R9-serien, kunna spela upp 1080p-filmer samt förbättrat grafikminneshantering.

Priset är cirka 30 brittiska pund. Samtidigt släpps en mindre uppdatering (1.2) gratis för ägare av 1.0-drivrutinen.
https://www.amistore.net

Bli en stjärna

Lär dig programmera i Hollywood

Av Samuli Holopainen. Introduktion av Johannes Genberg
Översatt av Johannes Genberg och Stefan Blixth

Har du någonsin velat lära dig programmering men tyckt att tröskeln varit för hög eller läroböckerna för komplicerade? Då är denna nybörjarguide för dig.

Introduktion

Hollywood är ett Lua-inspirerat programmeringsspråk, menat för grafiska presentationer. Men det kan också användas till projekt som normalt associeras med andra programmeringsspråk, som MUI-baserade nyttoprogram eller spel. Det finns till alla Amigor (Classic, AmigaOS 4, MorphOS och AROS) och kan kompilera till alla andra populära OS. Det är dessutom ett relativt enkelt språk, lämpligt för absoluta nybörjare.

Problemet är dock själva lärokurvan. Oavsett hur enkelt ett programmeringsspråk är så är det fortfarande en utmaning att lära sig hur kod fungerar rent logiskt. Det finns visserligen gott om nybörjarkurser i diverse språk, men min erfarenhet är att de oftast riktar sig till folk som redan har en bakgrund i programmering. Först ett enkelt ”Hello World” och sen snabbt vidare till mer avancerade saker som den redan vane tycker är intressant. Vi som aldrig skrivit en rad kod förväntas ändå hänga med och känner oss dumma för att vi inte klarar av andra kapitlet. Boken läggs ner och ännu en potentiell programmerare har självmant slutat i förtid, helt i onödan.

”Lära genom att koda” är det förslag jag hört till leda. När jag då kontrat med ”men jag kan inte ens den enklaste kod” får jag höra ”lär dig genom att läsa andra personers kod”. Det är lika hoppfullt som att börja lära sig japanska genom att läsa romaner.

Kod är kanske inte en ogenomtränglig djungel, men den är allt annat än självklar.

Genom mina egna erfarenheter (eller brist på dem på grund av ovan nämnda anledningar) föddes idén om att göra en ny sorts nybörjarguide. En som förutsätter att läsaren inte vet något om kodning. En som visserligen lär genom att visa exempel, men som också förklarar vad de olika delarna faktiskt gör. Läsaren ska aldrig känna att något viktigt hoppas över. När man nått slutet av artikeln ska varje del, från det enklaste till det mindre enkla, kännas logiskt. Samuli Holopainen var vänlig nog att ställa upp och skrev den nybörjarguide jag alltid själv velat läsa.

Förhoppningsvis kommer denna guide få fler att börja skriva egna program och spel till Amigan. Både för att vi behöver detta, men än mer för att det är så kul att vara kreativ.

Mål och medod

Detta är en nybörjarguide i Hollywood för de som inte har någon som helst erfarenhet i programmering. Istället för den traditionella metoden för att lära ut kommer denna guide förklara genom att visa hur man skapar ett enkelt PONG-spel.

Det enda du behöver är en kopia av Hollywood och en texteditor. Vissa texteditorer har inbyggt stöd för Hollywood och kan både rada upp koden på ett överskådligt vis samt köra programmen direkt, utan att man först måste kompilera

dem. Jag tänker inte förklara hur man använder Hollywood. Det utgår jag ifrån att du redan vet eller kan lista ut själv (läs manualen annars).

Ibland kommer kodraderna vara numrerade i början. Detta gör vi för att lättare kunna hänvisa till koden när vi förklarar dem, eller när den är för lång för att få plats på bredden. Detta är för att du inte ska tolka en rad som två (eller flera). Dessa nummer ska aldrig skrivas ut. Gör du det kommer programmet inte att fungera.

Om en rad börjar med "→" betyder det att det är det du kommer se när du kör ditt färdiga program.

Våra första program

Vi ska börja med klassikern bland nybörjarkod. Det så kallade "Hello World"-exemplet:

```
print(" Hello World" )
→ Hello World
```

Detta program har bara ett kommando (*command*): "print". De flesta Hollywood-program har också "()" i slutet av ett kommando: i detta fall "print()".

Denna "()" är delen där programmeraren kan – ja, i många fall rentav måste – lägga in ändamål för kommandona. I vårt exempel ovan används det för att säga åt skrivarkommandot "print" vad det är den ska skriva: "Hello World".

Du kommer nu märka att denna kod har ett problem. När du startar ditt program hinner du inte se någon text. Detta är för att samma mikrosekund som texten skrivs ut så avslutas programmet, då inget annat har angivits. Om du vill beundra ditt verk kan du skriva detta:

```
print(" Hello World" )
REPEAT
FOREVER
```

När du kör detta program kommer texten finnas kvar tills du trycker "CTRL-C", vilket är Hollywoods sätt att stänga av alla program.

REPEAT och FOREVER-kommandona är ganska självförklarande för den som kan engelska. Vi kommer återkomma till REPEAT senare.

Tills vidare kommer jag istället använda mig av "wait(50)" istället. Vad detta gör är precis som det låter: vänta en viss period innan programmet går vidare till nästa uppgift (vilket i slutet på ett program är av avsluta det). Numret efter "wait" är andelen tidsenheter eller "ticks", i vilka det går 50 per sekund. Alltså betyder "wait(50) att programmet ska vänta exakt en sekund. Vill du att den ska vänta längre skriver du bara ett högre nummer. Du kan också förstås använda REPEAT FOREVER.

Istället för att använda "print" kan du använda "debugprint". Det denna gör är att skriva text i till exempel ett annat fönster, vilket inte försvinner förrän du stänger det. Detta är användbart när man vill hålla koll på eventuella problem i programmet. Faktum är att innan jag skrev denna guide har jag alltid använt detta kommando istället för "print" då den alltid varit tillräcklig för mina behov.

Skillnaden är dock att "debugprint" kan se olika ut beroende på operativsystem. Genom att använda "print" istället kan jag försäkra mig om att läsarnas resultat är identiska. Märk väl att när man använder "debugprint" så behöver man inte använda "\n" (vilket betyder "radbrytning", mer om det senare) utan radbrytning sker då automatiskt.

Som du säkert märkt använder jag citattecken i mina print-kommandon när jag vill förklara vad jag vill skriva ut. Tar du bort dem så fungerar inte programmet längre. Men om du slår ihop de två orden i exemplet ovan händer något ännu konstigare:

```
print(HelloWorld)
wait(50)
```

Nu får du se det här:

```
→ NIL
```

Det är för att nu försöker ”print” skriva ut en variabel (*variable*) och inte en text. Nu undrar du förstås vad en variabel är.

Du kanske kommer ihåg matematiken från skolan när man skulle lista ut X i formeln ”2+X=3” etcetera. X fungerar som en typisk programmeringsvariabel. Om vi istället skriver ”2+X=?” så kommer vi ännu närmare normal programmeringspraxis. Beroende på vad X är får vi olika resultat. Det kan vara vilket nummer som helst, och ekvationen kan förändras allteftersom programmet arbetar och få nya, annorlunda resultat. Tanken med variabler är att de kan kan innehålla information. Både siffror och bokstäver.

Till exempel kan variabeln vara ”4”. Det kan också vara ett namn, som ”Andreas Falkenhahn”.

Variabler är användbara. Föreställ er att Amiga Inc. skriver alla sina brev med VD:ns fulla namn varje gång. ”Bill McEwen från Amiga Inc. hälsar ...”, ”Amiga Inc:s VD Bill McEwan har glädjen att meddela ...” och så vidare. Vad händer om VD:n plötsligt byts ut? Om alla brevmallar skrivs manuellt måste de skrivas om en och en. Om vi låter namnet vara en variabel X, och låter varje mall leta efter X på ett ställe istället sparar man mängder av arbete. ”X från Amiga Inc. hälsar ...”, ”Amiga Inc:s VD X har glädjen att meddela ...”. Vi kan alltså ändra namnet på hundratals mallar genom att bara byta ut X mot ”Trevor Dickinson” på ett enda ställe.

Så här kan en variabel se ut i Hollywood:

```
X = ”Hello World”
print(X)
wait(50)
→ Hello World
```

Det blir exakt samma resultat som i vårt första program, men medan vårt första program sade åt oss att skriva en specifik text så säger denna åt oss att skriva X, oavsett hur den ser ut.

En variabel kan vara vad som helst och se ut hur som helst, så länge man inte använder vissa speciella tecken eller ord som redan används internt av Hollywood (till exempel ”print” kan inte vara en variabel då den redan används som ett kommando). Det kan vara ”hillbillybilly”, ”vd” eller ”agentx” eller vad du vill:

```
vd = ”Bill McEwan\n”
agentx = ”James Bond\n”
hillbillybilly = 1
print(vd)
print(agentx)
print(hillbillybilly)
wait(50)
→ Bill McEwan
→ James Bond
→ 1
```

Tänk på att kalla dina variabler något enkelt och uppenbart, för din egen skull.

Nu kanske du har märkt att vi använt ”\n” i slutet av vissa texter. Det är som vi nämnt ”radbrytning”. Alltså ny rad. Det beror på att det inte sker automatiskt. Om vi till exempel skriver:

```
print(”Första raden”)
print(”Andra raden”)
wait(50)
→ Första radenAndra raden
```

Vi måste alltså själva ange var vi vill att det ska bli ny rad genom att använda ”\n” (som du kanske kommer ihåg behövs det inte om du använder ”debugprint”). Du kan lägga in detta var du vill och hur många gånger du vill:

```
print(”Första raden\n\nAndra raden”)
wait(50)
→ Första raden
→
→ Andra raden
```

Du kan även ändra variablerna allteftersom. Annars hade man inte kunnat göra mycket mer

än att definiera vad som ska bytas ut mot vad:

```
text = ”Hello World\n”
print(text)
text = ”En annan text\n”
print(text)
text = 123
print(text)
wait(50)
→ Hello World
→ En annan text
→ 123
```

Än så länge har vi använt ”print” för att skriva ut en text i taget. Du kan faktiskt skriva ut flera:

```
text1 = ”Hello”
text2 = ”World”
print(text1..text2)
wait(50)
→ HelloWorld
```

Genom att skriva ”..” mellan två variabler kan man lägga hur många sådana efter varandra man vill. Men här har det uppstått ett vanligt misstag bland programmerare. Som ni ser sitter de två orden ihop. Det är för att vi har sagt åt programmet att skriva ut de två texterna direkt efter varandra, utan mellanrum. Detta kan man förstås enkelt fixa.

```
1: text1 = ”Hello”
2: text2 = ”World\n”
3: print(text1..text2)
4: print(text1..”Jag glömde mellanrummet”..text2)
5: print(text1..” Nu kom jag ihåg det ”..text2)
6: print(text1..” ”..text2)
7: wait(50)
→ HelloWorld
→ HelloJag glömde mellanrummetWorld
→ Hello Nu kom jag ihåg det World
→ Hello World
```

Som ni ser på rad 4 så har jag lagt till egen text, men då jag inte lagt till något mellanrum sitter de ihop. Detta lade jag till på rad 5, mellan citattecknen och första respektive sista ordet. På rad 6 skrev jag inget annat än just mellanslag mellan citattecknen.

När variabler innehåller siffror kan de användas för matematiska ekvationer (och då använder man inte citattecken):

```
1: mysalary = 10000
2: wifesalary = 20000
3: totalsalary = mysalary + wifesalary
4: print(”Min och min frus sammanlagda lön är ”..totalsalary..” kronor per månad”)
5: wait(50)
→ Min och min frus sammanlagda lön är 30000 kronor per månad
```

Att tänka som en dator

Som du säkert redan listat ut så utförs kod en rad i taget, uppifrån och ner. Det du hänvisar till i koden måste alltså redan varit skrivet. Om vi tar ovanstående exempel kanske du tycker att det vore enklare för dig om du har variablerna sist. Pröva att lägga rad 1 och 2 efter rad 4 och se vad som händer:

```
→ 0
```

En människa kan lätt tolka vad det är du vill göra. Men en dator kan inte tolka, den kan bara läsa exakt som det står skrivet. När datorn försöker räkna ut summan på rad 3 (som nu är rad 1) så förstår den inte att den kan hitta siffrorna längre ner. Istället så räknar den ut summan baserat på ”mysalary” och ”wifesalary” kända värden. Det vill säga 0. Alltså, noll plus noll är lika med noll.

Fast det är inte riktigt sant. I själva verket är det en variabel utan information.

Man kan också säga att den inte har några värden. Man kan dessutom också säga att den

söker efter värden som inte blivit känt. Det är alltså inte egentligen ”0” utan en ”NIL”. Dessa två är snarlika men med en viktig skillnad. NIL betyder att det inte finns några värden alls. 0 är detsamma som ingenting, men är i sig självt ändå ett värde.

När NIL uppstår i matematiska kalkyler behandlar programmet det ändå som nummer ”0”. Det är lite överkurs, men tänk på att om ditt program buggar och ger dig resultatet ”0” så kan det egentligen betyda NIL. Alltså att det är fel i koden någonstans.

Det är också därför som ”print(helloworld)”-exemplet i början resulterar i ett NIL. Den letar efter variabeln ”helloworld”, men då den inte finns någonstans före ”print” resulterar det i att kalkylen saknar värden; alltså NIL. Det förra programmet har samma problem, men då det är en matematisk kalkyl ger den NIL värdet ”0”. Eller för att formulera det matematiskt: NIL + NIL = 0.

Man kan tänka sig att exekvera ett program är som att följa en vägbeskrivning som till exempel säger åt dig att: 1. sväng vänster, 2. sväng höger, 3. kör rakt fram så är du framme vid målet.

Om du då istället väljer att svänga höger på första punkten och sedan vänster kommer du säkerligen inte hamna på samma slutdestination. Detta gäller även vid programmering.

Har du inte tidigare tilldelat variabeln ett värde kan du inte anta att datorn ska fixa det hela automagiskt åt dig senare. Variabeln kan ju även komma att innehålla olika värden beroende på vart i koden den befinner sig. Det är du som styr datorn med ditt program från punkt A till punkt B, inte tvärt om.

Statisk och dynamisk kod

Så långt har programflödet varit rätt så självförklarande. Exekvera rad 1 sedan 2 och 3 och så vidare som sen till slut kommer vi till slutet av programmet.

Om vi nu ska använda en variabel som den egentligen är tänkt för och inte bara ha den som en container för statiska strängar och värden så ska vi börja se den som en dynamisk lagringsplats istället. Nu tänker du kanske, vad är skillnaden mellan statisk och dynamisk när vi pratar om variabler?

Nånting som är statiskt är ju som bekant nånting som alltid är samma och om vi tar vårt första exempel : ”print(”Hello World”)” så ser vi att det som skrivs ut alltid kommer resultera i samma sträng.

En dynamisk version av detta anrop skulle se ut som följande :

```
text = ” Hello World”
print(text)
```

”Print(text)” är en dynamisk operation då den är beroende av vad som för tillfället gömmer sig i ”text”-variabeln och som i slutändan kommer vara det som skrivs ut på vår skärm.

Om man ska dra en parallell för en källkod till ett spel så är där allt i princip uppbyggt naturligt dynamiskt beroende på spelarens val och vart den ska visas på skärmen.

Det mest grundläggande och vanligaste sättet att kontrollera flödet i en kod är ”IF”-satsen.

Ett enkelt ”IF” efterföljs av ett ”THEN”-uttryck som följande exempel visar :

```
1: A=1
2: IF A=1 THEN text=” Hello World”
3: IF A=2 THEN text=” Bye World”
4: print(text)
5: wait(50)
→ Hello World
```

I detta program väljer vi att sätta värdet ”1” på variabel A.

Andra raden kontrollerar om A är lika med 1 (vilket det är) och om så är fallet så sätts innehållet på variabeln ”text” till ”Hello World”. Tredje raden har samma kontroll fast jämför A med 2 (vilket det inte är) och hade det varit det så hade variabeln ”text” fått innehållet ”Bye

World". Fjärde raden skriver ut variabeln "text".

Om vi nu skulle vara busiga och byta ut första raden från A=1 till A=2 istället så skulle vi få:

```
→ Bye World
```

Och om vi ändrar variabel A till nånting helt annat så skulle vi få :

```
→ NIL
```

Detta beroende på att variabeln "text" inte har initierats med nått innehåll. Variabeln "text" får ett innehåll om någon av rad 2 eller 3 har uppfyllts i vårt program.

I alla andra fall kommer datorn skriva ut det enda värde som den då känner till. Vilket är "NIL".

IF – THEN-uttrycket är den enklaste formen av IF-satsen, men i många fall kan det finnas behov av att göra flertalet kommando än bara ett. Istället för att skriva följande:

```
IF A=1 THEN text = " Hello World"
IF A=1 THEN B=3
IF A=1 THEN print(text)
```

så kan vi istället skriva på följande sätt:

```
IF A=1
text=" Hello World"
B=3
print(text)
ENDIF
```

I ovanstående exempel så kommer alla rader/kommandon under "IF A=1" utföras om A är lika med 1, till och med raden ENDIF.

I vårt första "IF"-exempel så hade vi två stycken olika IF–THEN-satser och de berodde på tillståndet av variabel A. Det finns ett annat sätt som vi kan göta detta på.

Det är IF–ELSEIF–ELSE-satser och de ser ut på följande sätt:

```
1: A=2
2: IF A=1
3: text=" Hello World"
4: ELSEIF A=2
5: text=" Bye World"
6: ELSEIF A=3
7: text=" Another World"
8: ELSE
9: text=" A:s värde måste vara mellan 1 och 3"
10: ENDIF
11: print(text)
12: wait(50)
→ Bye World
```

Om man ändrar värdet på variabel A så kan man få 4 olika texter att skrivas ut. Detta fungerar på följande sätt:

Rad 2 är första gången koden kollar om värdet i rad 1 stämmer överens eller inte. Stämmer det går den vidare till rad 3 och utför det som står där. I detta fall ger den variabeln "text" värdet "Hello World", vilket ännu inte är utfört. Men då den inte stämmer går programmet vidare till rad 4, där det finns en ELSEIF. Den stämmer, så värdet för "text" blir "Bye World" istället. Nu hoppar programmet vidare till rad 6 där den hittar en annan ELSEIF. Men då det redan finns ett korrekt värde behandlar den ELSEIF som en ENDIF istället och går direkt till rad 10, oavsett hur många ELSEIF det finns innan dess. Därefter går den vidare till rad 11, där "text" skrivs ut.

Programmet kommer gå igenom alla ELSEIF tills den hittar ett korrekt värde. Men om det inte finns något sådant har vi en ELSE där också. Om vi då byter värde på rad 1 till vad som helst förutom 1, 2 eller 3 kommer "text" att få värdet "A:s värde måste vara mellan 1 och 3" och skriver detta istället. På detta sätt kommer detta program alltid att skriva något. För ELSE används ifall inget av de ovanstående värdena stämmer. Glöm bara inte att om du använder ELSE måste du ha en ENDIF för att stoppa det.

Till skillnad från IF–THEN så kan du använda hur många variabler och värden du vill i IF–ELSEIF–ELSE-koder:

```
A=1
IF A=1
text=" Hello World\n"
B=1
ELSEIF A=2
text=" Bye World\n"
B=2
ELSEIF A=3
text=" Hello World\n(again)\n"
ELSE
text=" A är felaktig\n"
ENDIF
IF B=1
mertext=" B är 1"
ELSEIF B=2
mertext=" B är 2"
ELSE
mertext=" B är felaktig"
ENDIF
print(text)
print(mertext)
wait(50)
```

Som du ser så kommer B få variabeln "1" ifall A=1. Är A=2 blir också B=2. Om A=3 finns det inget värde för B och då kommer det stå "B är felaktig" eftersom den inte har något värde och ELSE används. Om A inte är 1, 2 eller 3 kommer båda ELSE-texterna skrivas ut istället.

Du kan välja om du vill använda ELSE eller ELSEIF. Du kan använda båda eller inga om du vill. Du kan skriva IF–ENDIF lika väl som IF–ELSEIF–ELSE–ENDIF-satser. Så länge de används korrekt förstås.

Du kan också använda IF-argument inuti andra IF-argument:

```
A=1
B=1
IF A=1
IF B=1
text=" Hello World"
ELSE
text=" Bye World"
ENDIF
ENDIF
print(text)
wait(50)
→ Hello World
```

Om du ändrar B till "2" kommer du få "Bye World" istället. Ändrar du A kommer du få NIL, och B kommer aldrig exekveras.

Tänk om vi vill skriva ut tre olika texter efter varandra i en ändlös loop? Kan vi inte skriva såhär då?

```
A=1
IF A=1
text=" Hello World\n"
A=2
ELSEIF A=2
text=" Bye World\n"
A=3
ELSEIF A=3
text=" Another World"
A=3
ENDIF
print(text)
wait(50)
```

Nu borde väl alla tre skrivas ut efter varandra i oändlighet? Nej, faktiskt inte. Det som händer är att programmet kollar om A=1, vilket det ju är redan på andra raden. Därefter ändrar den visserligen variabeln till A=2 men då den har hittat rätt hoppar den direkt till ENDIF och fortsätter programmet:

```
→ Hello World
```

Det är exakt samma sak om vi skriver såhär:

```
A=1
```

```
B=1
IF A=1
text=" Hello World"
ELSEIF B=1
text=" Bye World"
ENDIF
print(text)
wait(50)
→ Hello World
```

Varför då? Jo, för att så fort den hittar en variabel som stämmer (A=1) så hoppar ju programmet direkt till ENDIF och fortsätter därifrån. Vad B då har för variabel blir irrelevant då den står efter en korrekt variabel.

Det är så IF–ELSEIF–ELSE–ENDIF fungerar. Den letar efter det första värdet som stämmer och hoppar sen omedelbart till slutet. Som vi nämnde tidigare så kan en dator till skillnad från en människa inte tolka. Den utför bara koden i rak ordning.

Fram tills nu har våra kodexempel alltid också utförts i rak ordning. De kanske hoppar över ett par steg genom IF-argument, men de har ändå alla jobbat ner mot den sista kodraden, och tvingar programmet sen att stanna. Alla koder utförs endast en gång. Det finns förstås program där detta är meningen, men andra program – inte minst spel – ska ju utföra samma kod om och om igen tills spelaren själv väljer att sluta använda det.

Repeat until ...

Det kan vi till exempel göra genom att använda REPEAT–UNTIL/FOREVER-argument:

```
A=1
REPEAT
IF A=1
text=" Hello World"
A=2
ELSEIF A=2
text=" Bye World"
A=3
ELSEIF A=3
text=" Another World"
A=1
ENDIF
print(text.." \n" )
FOREVER
→ Hello World
→ Bye World
→ Another World
→ Hello World
→ Bye World
→ ...
```

När ett program ser REPEAT kommer den markera platsen. När den sen ser FOREVER kommer den återgå till REPEAT och utföra koden igen. I detta fall i en ändlös cykel. Första cykeln så letar den efter A=1 och ändrar värdet till 2. Sen hoppar den till ENDIF och fortsätter. Den ser FOREVER, hoppar tillbaka till REPEAT och utför ELSEIF A=2 och så vidare och så vidare. Vid sista ENDIF blir "A" återigen "1" och på så sätt bryts inte programmet förrän användaren trycker CTRL-C (som stänger alla hollywood-program).

Om vi inte vill att programmet aldrig ska ta slut kan vi använda UNTIL istället:

```
A=1
REPEAT
A=A+1
print(" Hello World\n" )
UNTIL A=3
print(" Slut" )
wait(50)
→ Hello World
→ Hello World
→ Slut
```

När programmet börjar ser den REPEAT, lägger till 1 på "A", skriver ut "Hello World" och då A inte är lika med 3 hoppar den tillbaka och utför koden igen. Som du säkert märker så skriver koden ut "Hello World" bara två gånger

och inte tre. Det beror på en vanlig miss bland programmerare, även erfarna sådana. När koden börjar är A=1, men redan innan ”Hello World” skrivs ut har vi ju lagt till 1 på ”A”. A är alltså 2. Så nästa gång koden utförs har vi redan nått 3. Den skriver då ut ”Hello World”, noterar att ”UNTIL A=3” och hoppar då raskt vidare till nästa del, vilket är ”print(”slut”)”.

Precis som med IF kan du ha hur många REPEAT–UNTIL du vill inne i koden:

```
1: A=-1
2: B=0
3: REPEAT
4: A=A+1
5: B=0
6: REPEAT
7: debugprint(A..B.." \n" )
8: B=B+1
9: UNTIL B=10
10: UNTIL A=10
11: wait(50)
→ 00
→ 01
→ 02
⋯
→ 10
→ 11
→ 12
⋯ upp till 109.
```

Men vänta nu! Varför går koden upp till 109? Nu blir det lite komplicerat.

Notera att i rad 7 skrivs A och B direkt efter varandra, utan mellanrum. I rad 3 börjar REPEAT och där läggs det en etta på A i nästa rad och B fastställs som ”0” i raden efter. På rad 6 börjar ytterligare en REPEAT och nästa rad skriver ut A och B. På rad 8 läggs en etta på B. Det betyder att i första cykeln så blir A 0 och B också 0. Sen går den till nästa REPEAT som skriver ut dessa siffror (00) och lägger till 1 på B. Eftersom denna process har kommandot UNTIL B=10 på rad 9 så ska denna REPEAT fortsätta tills den når 10. Därefter går den vidare till nästa rad (10) där den stöter på en till UNTIL, som säger åt en att gå tillbaka till den första REPEAT (rad 3) och upprepa den processen tills den når 10. A får ytterligare en högre siffra och på rad 5 fastställs B som 0 igen (och inte 10). Därefter upprepar den REPEAT på rad 6 igen. Den når 10, hoppar till nästa UNTIL på rad 10 och det hela börjar om igen.

Eftersom vi skrivit B=B+1 på rad 8, efter ”print()” så kommer B aldrig upp i 10 innan UNTIL säger åt den att gå vidare, och sen nollställs igen på rad 5. Men eftersom det läggs på en etta till A på rad 4, och att på rad 6 går en till REPEAT-cykel för B så hinner programmet upp till 109 innan rad 10:s UNTIL ser att A=10 och avslutar programmet.

Detta betyder att 10-talen repeteras 11 gånger, medan entalen gör det 110 gånger (10 gånger 11). Räkna på det.

Notera att vi använde ”debugprint()” istället för ”print()”. Detta är för att försäkra oss om att du kan se alla siffror i ett shellfönter. Hade vi skrivet det med ”print()” hade troligen fönstret fått slut på plats när du bara nått halvvägs.

Nu har vi kommit så långt att du borde kunna experimentera med denna kod genom att lägga till IF-kommandon eller ännu fler REPEAT–UNTIL. Det finns ingen gräns hur mycket du kan lägga till. Eller ja, det skulle vara om RAM tog slut, men det är knappast realistiskt här.

Fram tills nu har våra exempel varit baserad på om jämförelser på värden stämmer överens eller inte. Men man kan också använda programmeringsmotsvarigheten till ”mer än” och ”mindre än”. Ta den här koden som exempel:

```
REPEAT
A=A+2
UNTIL A=10
```

Det fungerar så länge A adderas till 6, 8, 10 till exempel. Men vad händer om vi skriver A=1 ovanför REPEAT? Eller 11? Då kommer UNTIL

aldrig utföras eftersom vi aldrig kommer hamna på exakt 10. Istället kommer den cykeln att fortsätta i all evighet.

Därför skriver vi såhär istället:

```
REPEAT
A=A+2
UNTIL A > 10
```

”>” betyder ”mer än”. Om A då är 10, 11 eller 128 kommer UNTIL då räkna cykeln som avslutad. Naturligtvis betyder ”<” ”mindre än”. Men det tar vi en annan gång.

Dags att spela

Nu har vi kommit så långt att vi kan göra och förstå ett enkelt PONG-spel. Det kommer inte vara vackert eller optimalt skrivet, men det är förenklat så mycket som möjligt för den här guidens skull. Det finns också andra kommandon vi inte gått igenom än, men de är mycket enklare att förstå och behöver ingen djupare förklaring. Vi kommer ändå gå igenom dem allteftersom vi förklarar vad de olika delarna gör.

Observera att vi inte skriver ut tomma rader eller tomma ytor före kod på grund av platsbrist.

```
1: @SCREEN {Mode = ” FakeFull-
Screen” }
2: @DISPLAY {Width = 1920, Height =
1080, Borderless = True, ScaleMode
=  #SCALEMODE_AUTO, FitScale=True}
3: stickwidth = 40
4: stickheight = 200
5: playerx = 1800
6: playery = 500
7: computerx = 110
8: computery = 500
9: computerspeed = 3
10: ballx = 930
11: bally = 500
12: ballsize = 30
13: ballstartspeed = 1
14: ballspeed = ballstartspeed
15: ballspeedincrease = 1.10
16: ballxdirection = ” right”
17: ballydirection = ” up”
18: top = 0
19: bottom = 1050
20: leftedge = 50
21: rightedge = 1900
22: playerscore = 0
23: computerscore = 0
24: QUIT = 0
25: BeginDoubleBuffer()
26: REPEAT
27: Cls(#BLACK)
28: If ballxdirection = ” right”
29: ballx = ballx + ballspeed
30: If ballx > rightedge
31: ballx = 1000
32: bally = 500
33: ballspeed = ballstartspeed
34: computerscore = computerscore + 1
35: EndIf
36: ElseIf ballxdirection = ” left”
37: ballx = ballx - ballspeed
38: If ballx < leftedge
39: ballx = 1000
40: bally = 500
41: ballspeed = ballstartspeed
42: playerscore = playerscore + 1
43: EndIf
44: EndIf
45: If ballydirection = ” down”
46: bally = bally + ballspeed
47: If bally > bottom
48: ballydirection = ” up”
49: EndIf
50: Else
51: bally = bally - ballspeed
52: If bally < top
53: ballydirection = ” down”
54: EndIf
55: EndIf
56: If computery < bally
57: computery = computery + compu-
terspeed
```

```
58: ElseIf computery > bally
59: computery = computery - compu-
terspeed
60: Else
61: /*nothing*/
62: EndIf
63: playery = MouseY()
64: Box(ballx, bally, ballsize,
ballsize, #WHITE)
65: Box(computerx, computery,
stickwidth, stickheight, #WHITE)
66: Box(playerx, playery, stick-
width, stickheight, #WHITE)
67: TextOut(300, 30, computerscore)
68: TextOut(1630, 30, playerscore)
69: col = Collision(#BOX, ballx,
bally, ballsize, ballsize, com-
puterx, computery, stickwidth,
stickheight)
70: If col = 1
71: ballxdirection = " right"
72: ballspeed = ballspeed * ball-
speedincrease
73: EndIf
74: col = Collision(#BOX, ballx,
bally, ballsize, ballsize, playerx,
playery, stickwidth, stickheight)
75: If col = 1
76: ballxdirection = " left"
77: ballspeed = ballspeed * ball-
speedincrease
78: EndIf
79: Flip()
80: If IsRightMouse()=1 Then QUIT= 1
81: UNTIL QUIT=1
```

Rad 1 och 2 definierar vilket sorts skärm som ska öppnas. ”FakeFullScreen” betyder att programmet öppnar ett fönster som tar upp hela skärmen. Medan det fortfarande är ett vanligt fönster (du kan till exempel flytta runt det) så ser det ut som en skärm (skillnaden mellan fönster och skärm tänker jag inte ta upp här). ”@Display” beskriver hur ditt fönster ska bete sig. Till exempel att den har 1920x1080 i storlek och saknar fönsterramar (Borderless = True). ”ScaleMode = #SCALEMODE_AUTO, Fitscale=True” säger att den ska minska eller öka i storlek så den stämmer med din skärm. Vi behöver inte gå igenom dessa i detalj denna gång, kopiera dem bara rakt av.

Efter dessa har vi många variabler. Jag lägger dessa i början av programmet för att vi ska ha enklast möjliga överblick. Det är också lättare på så sätt att ändra på inställningar. Du kan till exempel ändra på värdet på ”ballspeed” och ”computerspeed”. Vissa variabler är dock delade. Det betyder att till exempel storleken på spelarens och datorns ”racket” använder samma värde och är därför identiska. Du kan inte ändra på storleken på ditt racket utan att datorns också får samma storlek.

Notera ”QUIT = 0” på rad 24. Det är här vi har lagt in en avstängningsfunktion för spelet, separat från CTRL-C.

Vi ser att den har värdet ”0”. Programmet slutar med andra ord inte när den läser denna rad. Därefter kommer en bekant REPEAT–UNTIL (rad 26), hela vägen ner till rad 81. Eftersom programmet går runt runt kommer den ständigt tillbaka hit. Fram tills UNTIL QUIT=1. Precis före detta har vi ”isrightmouse()”. Är den ”0” betyder det att höger musknapp inte är nertryckt. ”1” betyder att den är det. Så är höger musknapp nertryckt blir värdet på QUIT ”1”. När det händer, då stängs programmet ner när den når rad 81. Annars åker programmet tillbaka till rad 26.

Nästa del är rörelsemomentet. I detta spel har vi gjort det väldigt enkelt: bollen rör sig i fyra olika riktningar. Upp och vänster, upp och höger, ner och vänster eller ner och höger. Detta är uppdelat i två olika delar där den ena delen definierar om den går upp eller ner och den andra om den går till höger eller vänster. Vi använder ”ballxdirection” och ”ballydirectrion” för att definiera dessa.

Titta på rad 28 och 29. Där står det att om

bollen åker till höger ska vi öka dess ”ballx”. För att förstå den här biten måste vi förklara en sak angående skärmar.

Om vi till exempel har en skärm på 1920x1080 så betyder det att det finns 1920 skilda punkter på den vertikala x-axeln. Alltså är punkt 1 längst till vänster och punkt 1920 längst till höger. Så ju högre siffra, desto mer åt höger. Y-axeln fungerar på liknande sätt. 1 är då högst uppe och 1080 är längst nere. Alltså är ”1,1” högst uppe till vänster och ”1920,1080” längst nere till höger.

Så om jag skriver ”ballx = ballx + 1” betyder det att bollen flyttar sig en punkt (eller en pixel) åt höger. Men som du säkert märkt så använder vi inte detta utan ”ballx = ballx + ballspeed” istället. Detta är för att lättare kunna bestämma hastigheten på bollen under spelets gång. När spelet börjar är hastigheten ”1” (se rad 13). Om vi skriver ”ballx = ballx + 1” så ökar hastigheten med exakt 1 varje gång den ska öka i hastighet. Genom att skriva som vi gjort kan vi öka hastigheten relativt istället för absolut, vilket ses på rad 15 (”ballspeedincrease = 1.10”). Ett annat sätt att räkna är att bollens hastighet ökar med 10%.

Nu är bollen i rullning. Nästa steg som programmet kollar är om den nått högra kanten än. Se rad 30 till 35. ”Rightedge” definierades på rad 21. Så rad 30:s kod säger att ”om bollens x-läge är på 1900 så stämmer IF”. Därefter hoppar bollen till x-läge 1000 och y-läge 500 på skärmen (ungefär mitten av skärmen), bollen får samma hastighet som ”ballstartspeed” (ursprungshastigheten) och datorn får en poäng. Alltså; du missade bollen.

Om du tittar på rad 36 till 43 kan du enkelt lista ut att det är ungefär samma sak där, fast tvärtom istället.

Nu tar vi en titt på rad 45 till 55. Som du ser är koden lite annorlunda här. Först kollar programmet om bollen går neråt (y-axeln). Ifall den gör det beter den sig på samma sätt som x-axeln. Sen kollar programmet om den nått botten. Gör den det så åker den tillbaka upp igen (istället för att hoppa till mitten och ge någon poäng). Likadant är det med koden under, fast tvärtom förstås.

Därefter kommer en mycket enkel AI för datorn (rad 56 till 62). Det datorn gör helt enkelt är att försöka matcha var bollen är för tillfället. Om bollen är högre än datorn racket åker den ditåt, och tvärtom.

Notera ELSE-kommandot. Först ser vi ”/*”, följt av ”nothing” och avslutas med ”*/”. Det är en kommentar. Den kommer inte läsas av programmet utan helt enkelt ignoreras. Detta skriver programmeraren för att hålla koll på vad han eller hon gör. Gör det till en god vana att skriva kommentarer. Annars riskerar du efter ett tag att inte kunna få en god överblick över ditt program utan måste lägga ner tid med att analysera vad du gjort istället.

Viktigare här dock är att i denna ELSE står ingenting. Det är för att det finns instruktioner om vad datorn ska göra om bollen är ovanför eller under racketen. Men om det är jämsides? Då ska den ju stå still istället. Det är det denna avsaknad av kod gör. Är bollen jämsides, gör ingenting. Faktum är att denna ELSE är onödig egentligen, men jag tog med den ändå i utbildningssyfte.

Nu ska vi titta till spelgrafiken (rad 64 till 68). Det är kort och gott två rektanglar och en kvadrat (datorns racket, spelarens racket samt bollen). Om du tittar på rad 3, 4 och 12 så står det där hur stora de ska vara. Vi använder också ”textout” för att skriva ut poäng. Det är inte mer komplicerat än att ”textout” skriver ut resultaten (”computerscore” och ”playerscore”) exakt där du sagt att den ska (på x- och y-axeln). Placeringen för racketen ser vi på rad 5 till 8. ”#WHITE” är förstås färgen på sakerna. Den koden är ett så kallat ”reserverat ord”, denna för färgen vit.

Som du ser så står ”ballsize” två gånger på den första ”Box”. Det beror på att det är en kvadrat så vi kan använda samma värde två gånger. Ändra värdet på rad 12 om du vill. Först definierar vi var den ska vara någonstans och sen bredden

och höjden på bollen. Vi gör likadant med våra två racket, som också delar samma värde för båda spelarna.

Notera att ”computerx” och ”playerx” är statiska. De kommer aldrig förändras under programmets gång. Detta förstås för att vi vill att deras racket bara ska gå upp och ner (y-axeln). Därför står rad 63 som den gör. ”MouseY()” där innebär att spelaren racket följer musens rörelser i y-axeln.

Nu ska vi kolla om bollen har träffat spelarens eller datorns racket eller inte (rad 69 till 78). Kommandot ”col” är helt enkelt en variabel som kollar resultatet av ett kollisionstest (vi döpte den till ”col”, men den kan heta vad som helst). ”0” är ingen kollision förstås, medan ”1” är det.

Det första ”Collision” kollar vad det är för något som ska kollidera, vilket är någon ”#BOX” (rad 64 till 66). Därefter definierar vi placeringen på x- och y-axeln på den första ”boxen”. Därefter dess bredd och höjd vilket ju är samma värde (notera att det är bredd först och höjd sen precis som det är x-axeln först och y-axeln efter. De korrelerar med varandra med andra ord). Därefter datorns rackets x- och y-position följt av dess bredd och höjd. Detta upprepas sedan på andra halvan, fast tvärtom då det är spelarens värden som ska definieras.

Tanken här är att om något av dessa värden träffar varandra så räknas det som en kollision. I teorin skulle vi kunna förvänta oss att bredden och höjden på bollen eller racketen träffas och räknas som en kollision, men det kan aldrig hända i praktiken eftersom höjd och bredd aldrig glider in i samma område. På detta sätt har vi gjort det enkelt för oss genom att skriva så hät, eftersom vi vet att det i praktiken innebär att de enda som kan träffa varandra är bollen samt datorns och spelarens racket. Vilket ju är poängen.

Gör den det så blir ”col = 1”, och bollen skiftar riktning åt motsatt håll. Eftersom detta är ett mycket enkelt spel är detta allt vi behöver göra.

Sist så har vi bestämt att varje gång bollen träffar ett racket så ökar hastigheten på bollen. Vilket syns på rad 15 som 10%. För varje träff blir det ytterligare 10% ökning på det existerande värdet. Hade vi skrivit ”2” istället för ”1.1” hade det inneburit en 100% ökning vid varje träff. Alltså 100% till 200% till 400% och så vidare.

Det sista vi ska förklara är ”Cls (#BLACK)” (rad 27), ”BeginDoubleBuffer()” (rad 25) samt ”Flip()” (rad 79).

”Cls” betyder ”clear screen” och rensar alltså skärmen. I vårt fall säger vi åt den att skriva ut svart färg varje gång den rensar skärmen. Varför vill man göra det? Jo, för gör vi inte det så kommer spelet rita om sig över det gamla, utan att ta bort det först. Efter ett tag skulle vi bara se en massa vitt gytter. Det är för att det vi uppfattar som rörelse inte alls är det, utan bara skärmen som ritar om sig om och om igen vilket skapar illusionen av rörelse. Pröva att ta bort raden och se vad som händer.

Men det räcker inte med ”cls”. Utan ”BeginDoubleBuffer” och ”Flip” så skulle illusionen av rörelse inte fungera. Istället skulle du se skärmen bli rensad på innehåll och sen skulle programmet rita upp de olika innehållen ett och ett. På modern och snabb hårdvara kanske processorn orkar göra det så snabbt att du inte ser någon skillnad ändå (då programmet jobbar så fort den orkar), men troligen skulle det resultera i en massa jobbigt blinkande.

”BeginDoubleBuffer” måste skrivas innan grafikens REPEAT–UNTIL börjar eftersom detta kommando bara ska användas en gång. Det programmet gör här är att rita upp allt som händer i programmet först i bakgrunden och visa det först när allt är klart, istället för att rita upp alltefter-som. Det ”Flip” gör då är att visa det ”BeginDoubleBuffer” nu ritat upp. Du kan föreställa dig någon som antingen ritar på ett papper medan du ser på, vänder blad och gör samma sak igen. Eller någon som visar ett färdigritat papper och vänder inte blad förrän nästa sida också är färdigritat. Det är därför det heter ”Flip”; vända blad.

Då var vår PONG-klon färdig. Kompilera och spela det.

Det sista liten detalj

Som koden ser ut så är all rörelse baserad på cykler, och en cykels längd varierar beroende på hur snabb din dator är. Är den långsam kommer bollen ta en evighet innan den når ett racket. Är datorn snabb tar spelet slut innan du ens förstått att det startat. Du har säkert stött på problemet när man spelar riktigt gamla spel på moderna maskiner. Det startar, en massa saker händer jättefort och sen är spelet slut (och du har oftast förlorat). På det tiden resonerade man att ”ju snabbare dess bättre” eftersom spelen fungerade mjukt och smidigt på den tidens snabbaste hemdatorer. Vilket då bara var en bråkdel av vad en helt medelmåttig dator idag klarar av.

Idag resonerar man naturligtvis annorlunda, och vi vill att programmet ska vara lika snabbt på alla datorer. Det fixar vi genom att programmera in en klocka, och basera all rörelse på hur lång tid det tog för datorn att arbeta istället.

Först, skriv in detta direkt före REPEAT:

```
StartTimer(1)
```

Detta startar klocka nummer 1. Den fungerar som ett tidtagarur. Därefter skriver du detta precis innan UNTIL:

```
Repeat
timepassed = GetTimer(1)
Until timepassed > 3
ResetTimer(1)
timemultiplier = timepassed / 1000
```

Vi har nu skapat en REPEAT–UNTIL inom en existerande REPEAT–UNTIL som inte gör något annat än att räkna ner tills den kan gå vidare. Först så hämtat den värdet från ”StartTimer(1)” genom variabeln ”GetTimer(1)”, vilket också är 1 eftersom den letar efter klocka nummer 1. När fler än 3 tidsenheter passerat sedan klockan startade går den vidare och nollställer klockan igen. Som du minns så jobbar en dator utan klocka så snabbt den orkar.

Har du en någorlunda snabb dator märker du att bollen flyttar sig mycket långsammare nu än innan. Faktum är att den bara flyttar sig en pixel per sekund. Varför då?

Tittar du på rad 13 ser du att ”Ballstartspeed = 1”. Men detta värde är odefinierat och räknas därför som NIL. Spelet står alltså still (0 * 0 = 0 som du minns). Det är här ”timemultiplier = timepassed / 1000” kommer in i bilden.

”Timepassed” anropar ”GetTimer(1)” för första gången, men saknar alltså fortfarande värde. Lite längre ner ser vi ”timemultiplier = timepassed / 1000”. Det är här vi definierar värdet för klockan: i tusendels sekunder. Faktum är att Hollywood alltid räknar i mikrosekunder. Så 1000 är alltid detsamma som en sekund. Från och med nu blir ”ballstartspeed = 1”:s hastighet en pixel i sekunden. Spelaren kommer dock troligen aldrig hinna märka någon skillnad.

Okej, men vad är då poängen med koden?

Jo, först anropar ”timepassed” klockan och frågar hur lång tid det gått. Den får därefter veta att när den passerat 3 tidsenheter får den gå vidare. Annars ska den gå tillbaka och läsa samma kod igen tills den gör det. När klockan passerat 3 tidsenheter så nollställs den och redan i nästa rad definieras en tidsenhet som en tusendels sekund. Tiden gånger den passerade tiden delat med 1000.

För att minska på förvirringen så får du inte glömma att även om vi introducerar en klocka i systemet så arbetar fortfarande datorn så snabbt den kan för att läsa igenom koden. Det detta gör är att stanna upp koden i en loop, ungefär på samma sätt som att klicka på paus-knappen på ens fjärrkontroll regelbundet.

Men här har vi ännu ett problem.

Datorer kan läsa igenom kod olika snabbt beroende på hur kraftfulla de är. En långsam dator kanske inte har nått ”timepassed > 3” förrän efter 10 tusendels sekunder. Det betyder att den då

redan passerat ”3” med god marginal. Det innebär att programmet uppdateras vart 100-dels sekund. En snabb dator däremot kanske kommer hit på mindre än en tusendels sekund. Då loppar koden tills den passerat 3. Det kan vara ett ojämt nummer (som 3.01) men för enkelhetens skull säger vi att den hamnar på exakt 4. Detta gör att programmet uppdateras en gång per 250-dels sekund.

Uppdatering av koden har naturligtvis inget att göra med hur programmet uppfattar tid, vilket vi som sagt definierat i ”timemultiplier = timepassed / 1000”. En sekund är fortfarande en sekund oavsett hur snabb eller långsam datorn är. Det påverkar bara hur snabbt programmet tillåts arbeta.

Anledningen att vi har ”timemultiplier = timepassed /1000” efter ”ResetTimer(1)” är för att om den skulle vara före så skulle dess värde nollställas efter varje gång den kollar om den passerad 3 tusendelars sekunder. Vilket den riskerar att inte göra om datorn är snabb nog och då kommer den aldrig ur loopen.

Observera också att det är viktigt att ha ”ResetTimer” (som i den här koden nollställer klocka nummer 1) så snart som möjligt efter ”GetTimer”.

Vem vänta, vad är nu poängen med allt detta?

Jo, för nu använder vi det resultatet vi får genom denna metod att sakta ner programmet för att reglera hur snabbt resten av programmet ska arbeta. Skriv om dessa rader:

```
29: ballx = ballx + (ballspeed *
timemultiplier)
37: ballx = ballx  -  (ballspeed *
timemultiplier)
46: bally = bally + (ballspeed *
timemultiplier)
51: bally = bally - (ballspeed *
timemultiplier)
57: computery = computery + (com-
puterspeed * timemultiplier)
59: computery = computery  -  (com-
puterspeed * timemultiplier)
```

Det här är ganska enkelt. Om din dator är långsam och koden uppdateras per hundradels sekund (som vi nämnde ovan) så kommer bollens och datorns rackets hastighet att adderas med det samlade värdet för hur snabb bollen/racket är nu gånger tiden som gått (till exempel 1 + 1 * 0,01), men har du en snabbare dator som tvingas sakta ner och räkna i till exempel tvåhundrafemtiodelars sekunder så kommer bollens hastighet multipliceras med detta värde istället.

Det betyder alltså att hastigheten i spelet kommer vara identiskt för alla datorer (snabba såsom långsamma), då en sekund är en sekund är en sekund.

Men varför har vi ”Until timepassed = 3”? Räcker det inte med 0, eller 1? För det första, har vi noll är risken att en snabb dator kommer hinna komma hit på mindre än en cykel. I sånt fall nollställs inte klockan vid en användbar tidsram utan ger ett värde på 0. Det innebär att alla rörelser också multipliceras med noll. De står alltså still.

”1” är ingen bra siffra heller. Om en dator kommer hit på en halv cykel och en annan på 2/3-delar så innebär det att den ena fortsätter programmet när den kommit till exakt en tusendel medan den andra måste vänta tills den kommit till en och 333 tusendelars sekund. Alltså skulle programmen jobba med 33% skillnad på tidshållningen, vilket ganska snart skulle märkas. Därför väljer vi ”mer än 3”.

Detta är inte perfekt heller då det finns en risk att en dator fortsätter utföra kod efter 3,1 tusendelar och en annan efter 3,9 tusendelar. Men sannolikheten för det är mycket lägre och skillnaden skulle inte bli lika stor (max ungefär 25% skillnad). En alltför hög siffra riskerar å sin sida att göra att spelet inte fungerar smidigt på långsamma datorer. ”Mer än 3” är alltså en bra kompromiss.

Nu borde du kunna experimentera med koden och ändra på värden tills det passar dig bäst. Grattis, du har tagit ett stort steg att bli en programmerare! Svårare än så här är det inte.

Efter C64:an

Finns det en Generation 500 att berätta om?

Av Jimmy Wilhelmsson

I *Generation 64* fick läsaren träffa en mängd kända och halvkända underhållare och affärsmän som menade att C64:an var en anledning – kanske till och med största anledningen – till deras nuvarande karriärsval. Men hur är det med Amigan? Vilken roll hade den?

I mitt dagliga yrkesliv träffar jag många utvecklare, IT-chefer, spelkreatörer och entreprenörer – och påfallande ofta resulterar våra samtal i en nostalgisk och intressant diskussion om hemdatorn Commodore 64.

Samtalen behöver inte enbart handla om hur det var att utveckla på denna dator, vilket ju inte nödvändigtvis var mer speciellt än på någon annan dator, utan kunde handla om allt från känslan att ladda in och vänta på ett spel, hur demoscenen under en period var viktigare än skolan eller om hur det var att äga och hantera en hemdator i Sverige på 80-talet.

Som en del kanske vet resulterade alla dessa möten i boken ”Generation 64”, en kaffebordsbok om de unga svenskar som på 80-talet lät hemdatorn Commodore 64 påverka och förändra deras liv. 30 kända och okända svenskar intervjuas om sina helt olika livsöden som alla cirkulerar runt samma brungråa brödbox med 64 kilobytes under skalet.

Resan inleds lite försiktigt med svenska datorn ABC 80 och den tidige datorentusiasten Henrik Schyffert. Som avrundning låter jag Amiga 500 göra entré och Peter Sunde och Angelica Norgren får berätta om sina erfarenheter efter det att många svenskar förpassat sin C64 till garderoben till förmån för 16-bitarsdatorerna.

I berättelsen om Commodore 64 är samtliga personer ovan att betrakta som gästmedverkande – men om vi leker med tanken att resan sömlöst skulle gå vidare så fäster jag fokus vid framförallt två frågor.

Finns det en generation motsvarande Generation 64 att tala om på Amiga-plattformen? Om ja, mot vilken bakgrund skulle då denna berättelse kunna uppdags och presenteras?

Amiga-plattformen är, som ju alla i den här föreningen är väl medvetna om, inte en enhetlig sådan utan består av flera olika operativsystem och även flera olika hårdvaror. Medan specifikationerna för en Commodore 64 såg i stort sett likadan ut under tio års tid kunde även den enklaste av Amiga-modellerna, Amiga 500, relativt tidigt under sin livstid byggas ut med en halv megabyte mer RAM – vilket förändrade förutsättningarna för mjukvarutillverkare såpass att en del program inte fungerade på oexpanderade datorer eller åtminstone fungerade olika.

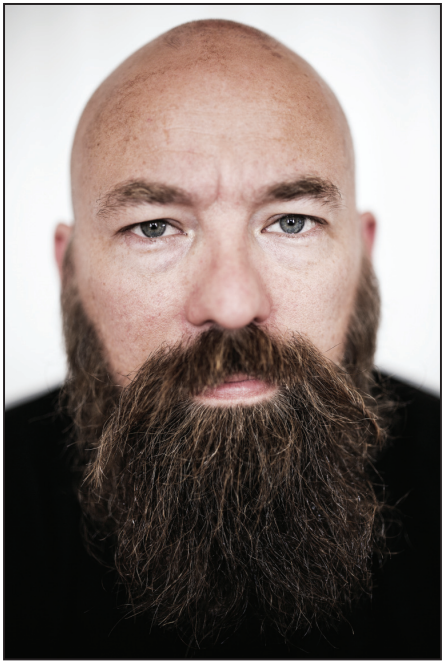
Ännu mer komplicerat blev det vid de tillfällen operativsystemet Workbench uppdaterades.

Inför bytet från Workbench 1.3 till Workbench 2.0 krävdes dessutom ett komponentbyte de flesta konsumenter inte kunde utföra själv. Att byta ut komponenter är på en stationär PC i dag ett ganska enkelt handgrepp men på en hemdator som Amiga 500 var det inte så.

Kortfattat; Amiga-plattformen är modernare än Commodore 64 på flera vis och hela systemstrukturen i både hårdvaru- och mjukvarumässigt var komplexare, närmare den verklighet vi lever i i dag. Det finns helt enkelt inte en enhetlig Amiga-plattform på samma vis som tidigare hemdatorer uppvisade.

Ännu mer kortfattat; finns Generation 500? Och vem är det isåfall?

I själva termen ”Generation 500” ingår implicit att det faktiskt i grunden handlar om en enda datormodell, Amiga 500. Detta är datorn flest unga förmodligen startade med för att sedan möjligen uppgradera och byta ut till större och modernare Amiga-datorer. Men generationen kan givetvis tolkas bredare, precis som Generation 64.



Fotograf: Lars Jansson

Men om berättelsen om Generation 500 till största delen skulle handla om datorn Amiga 500 – mot vilken bakgrund bör en sådan berättelse presenteras? Och hur? Är det återigen en bok som bör skrivas av någon, eller kanske något annat?

Vilken roll spelade Amiga-generationen för framväxten av Internet, fildelning och utvecklingen av den digitala omgivning vi i början av 1990-talet bara kunde skönja bortom horisonten?

Vad skilde Amiga-demoscenen från den klassiska C64-demoscenen, om något? De unga demomakare som startade med Amiga och inte var gamla nog att hoppa på Commodore 64 – på vilket sätt är deras resa annorlunda och vad hade det för betydelse, om någon?

Vad skänkte Amiga-generationen musikvärlden? Här pratar vi om en hemdator som kunde hantera samplingar på riktigt och som inte längre förlitade sig på analoga hårdvaruljud. Trackern, som vi nuförtiden känner den, föddes på allvar – och vem som helst kunde börja göra njutbar musik.

Vilken betydelse hade Amiga-grafikerna för den digitala bild- och animationsvärld vi lever i nu?

Vi började smått förstå att digitala bilder verkligen kunde mäta sig med de fysiska pappersbilderna – och Amigan blev bland annat många tv-stationers favoritalternativ.

Och slutligen – på vilket sätt påverkades tillväxten av en kodande befolkning av att det inte längre väntade en programmeringsprompt direkt efter påslagning?

Många frågor blir det – och kanske kan du hjälpa mig? Finns det en Generation 500 att tala om – och vad bör isåfall berättelsen innehålla? Fundera – och om du vill, hör av dig till mig på min hemsida via kontaktformuläret där.

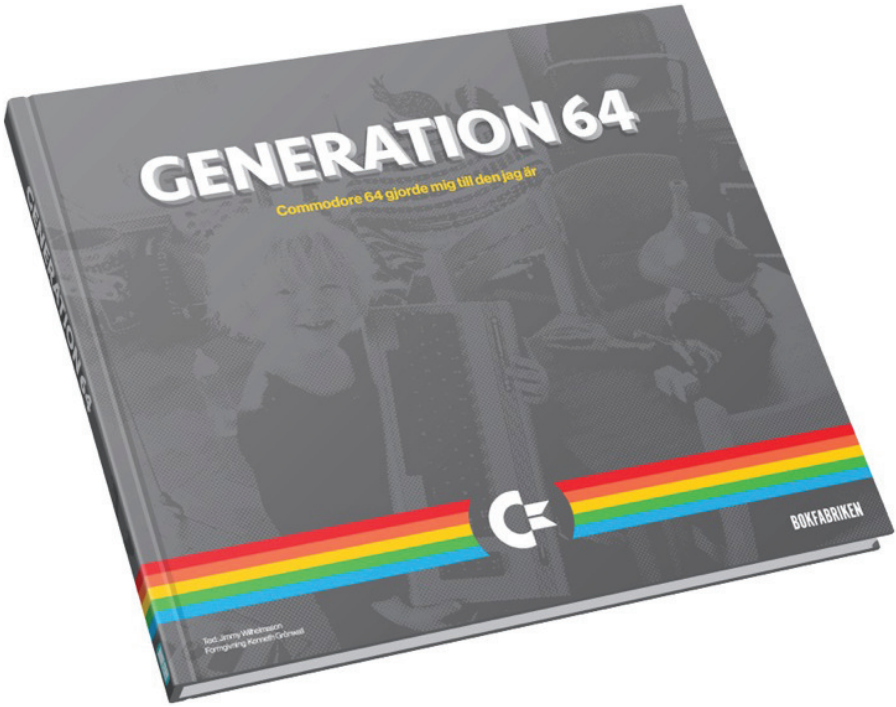
Det är inte säkert att Amiga-generationen behöver en egen bok; men det är säkert att frågan iallafall behöver stötas och blötas.

www.spelpappan.se

AMIGAstore.eu

Snabbtitten

Utgivare: Bokfabriken
ISBN: 9789187301834
Http://generation64.se
År: 2014
Författare: Jimmy Wilhelmsson
Pris: Cirka 298 kronor
Recensent: Johannes Genberg



Generation 64

Detta är en coffe table-bok full av historier från människor vars liv kom att formas av denna i Tyskland kallad ”folkdator”. Den är känd i Guinness rekordbok som världens mest sålda enskilda datormodell. Fortfarande.

Boken innehåller en samling historier från kända underhållare, affärsmän och internetrebeller. Dess hypotes är att Sverige kan tacka den här datorn för sin nuvarnade position i IT-världen.

Men det handlar mindre om hur fantastisk datorn i sig var och mer om vilka fantastiska saker den inspirerade andra att göra med den.

Inte bara bild och musik förstås, utan även mer ljusskygga saker som att lära sig cracka spel eller att sprida dem vidare till sina fattiga kamrater.

Detta är en medryckande, övergriplig och färgglad bok som har en given plats i bokhyllan för alla datorarkeologer.

Även Amigan får vara med i ett hörn. För många var det den naturliga efterföljaren. Commodore lyckades med den här datorn skapa en entusiastisk och trogen skara användare som följde med in i nästa generation. Men glöm inte att det var här det hela egentligen började.

ALINEA COMPUTER SHOP

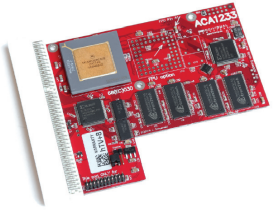
www.amiga-shop.net

The onlineshop for Classic and NG Amigas!

Now with new Webshop!

All Prices incl. 7/19% Tax excl. Shipping. The trademarks, names of products and company-names are belongings of its owners.

Hardware



ACA 1233/40 128 MB cer
A1200 turbo board with 68030
40 Mhz and 128 MB Ram
only 189,95 Euro

ACA 1220/33 128 MB... 134,90 Euro
ACA 1232 EC030/25 128 MB
..... 124,95 Euro
ACA500/14 2 MB.....79,95 Euro
Kickstart 3.1 Rom A2000 19,95 Euro
Kickstart 3.1 Rom A1200 19,95 Euro
Kickstart 3.1 Rom A500...19,95 Euro
Kickstart 3.1 Rom A600...19,95 Euro
Spider II USB 2.0.....89,95 Euro
Indivision AGA Mk2 cr
A4000/CD32 ..... 199,95 Euro
Indivision AGA Mk2 CR
A1200/4000T(*) ..... 159,95 Euro
Indivision ECS ..... 99,99 Euro
PC-Key 600.....39,95 Euro
A500 memory ext
with 512 KB.....28,95 Euro
A1000-Adapter f. Indivision ECS
.....12,90 Euro
AmigaOne keyboard (D).....29,95 Euro
TrueIDE.....39,90 Euro
Cocolino PS/2-Mousead. ....34,95 Euro

Arcade Evolution Amiga. 64,95 Euro
Arcade Evolution USB.....69,95 Euro
Dual CF-2,5 IDE Adapter ....16,95 Euro
CDTV remote control.....14,99 Euro
BVision KIT ..... 9,95 Euro
A604n memory ext.....34,90 Euro
Keyrah v2.....34,90 Euro

Plexiglas case for ACA500
Dust protector and it looks nicer
only 39,95 Euro

Software

Ace Of Hearts CD-Version
Video poker in a new dimension for
AmigaOS 4.1
only 19,95 Euro

AmigaOS 4.1 für A1/PII.....124,95 Euro
Freespace - The Great War 19,95 Euro
Freespace 2 (OS4.1).....19,95 Euro
M.A.C.E..... 29,95 Euro

Battle Squadron Collector's Edition DELUXE..... 30,00 Euro
AmiPhoto Downl..... 14,95 Euro
StormC5ED - CD Ver.....34,95 Euro
AmiWebView v2 CD..... 9,95 Euro
ANotice v2 - CD..... 9,95 Euro
StormC v4..... 49,95 Euro

Misc

Official AmigaOS beach boingball
Diameter: ca. 28cm/d
Must have for every Amiga fan
only 8,95 Euro

Official AmigaOS plush boingball
.....19,95 Euro
USB LED Fan..... 14,90 Euro
Floppy Disk Sticky Notes.....9,95 Euro
Sticker "Boing 300 mm".....14,95 Euro
Sticker "Boing 100 mm".....4,95 Euro
Magnet sticker "Boing".....4,95 Euro
Amiga pack of cards.....4,95 Euro
Out of coffee error - Tasse....6,95 Euro
ANNEX - Keep the Momentum Going..
.....7,95 Euro
Amiga Joystick-/ Mouse ext (1,8 m)
.....5,95 Euro
AmigaOne cup.....9,90 Euro
Retro Shirt.....12,00 Euro
Pac-Man cup.....9,95 Euro
Tetris ice cubes.....9,95 Euro
Tetris Heat Changing Mug....8,95 Euro
Ice Tray Binary Code.....9,95 Euro
Amiga Immortal 4.....24,95 Euro
AmigaOS Boing Poster.....11,00 Euro

Meine Erinnerungen an Commodore und Amiga(german book) 24,90 Euro
Die Commodore Story (gb)..12,95 Euro
Amiga - Quo vadis? (gb).....16,80 Euro
On the Edge: The Spectacular Rise
and Fall of Commodore.....28,95 Euro
VOLKSCOMPUTER(gb).....27,80 Euro

ALINEA COMPUTER

Geschwister-Scholl-Str. 26 • 61273 Wehrheim • Germany
Phone: (0 60 81) 5 76 99 88 • Fax: (0 60 81) 5 76 99 89
E-Mail: info@alinea-computer.de • http://www.alinea-computer.de

HOLLYWOOD

Hollywood is a multimedia-oriented programming language that can be used to create graphical applications very easily. It was designed with the paradigm to make software creation as easy as possible in mind. Thus, Hollywood is suited for beginners and advanced users alike. Hollywood comes with an extensive function library (encompassing over 600 different commands) that simplifies the creation of 2D games, presentations, and applications, to a great extent. It has been in development since 2002 and hence is today a very mature and stable software package.

One of the highlights of Hollywood is its inbuilt cross-compiler which can be used to deploy software on many different platforms without having to change a single line of the code. The cross-compiler can compile for all platforms from any platform Hollywood is running on. For instance, you can compile Mac OS X application bundles using the Windows version of Hollywood.

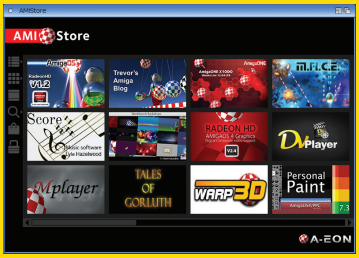
Hollywood is a light-weight, but still powerful programming language that is just about two megabytes in size and does not require any external components.

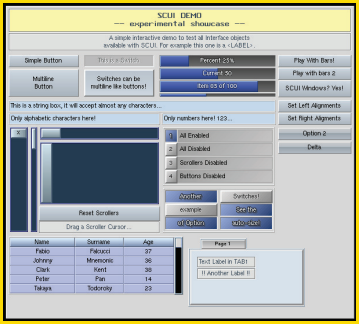
Hence, it is ideal for creating programs which run right out of the box. In fact, Hollywood programs will run perfectly from an USB flash drive without any prior installation whatsoever.

The following platform architectures are currently supported by Hollywood:

- AmigaOS 3 (m68k)
- AmigaOS 4 (ppc)
- Android (arm)
- AROS (i386)
- Linux (i386)*
- Linux (ppc)*
- Mac OS X (i386)*
- Mac OS X (ppc)*
- MorphOS (ppc)
- WarpOS (m68k/ppc)
- Windows (i386)

** Hollywood can save executables for Mac OS X and Linux but it is currently not available in a stand-alone version for these platforms. Thus, you currently have to use either the Windows or AmigaOS compatible version of Hollywood to save executables for Mac OS X and Linux.*





www.hollywood-mal.com