Examensarbete i Informatik

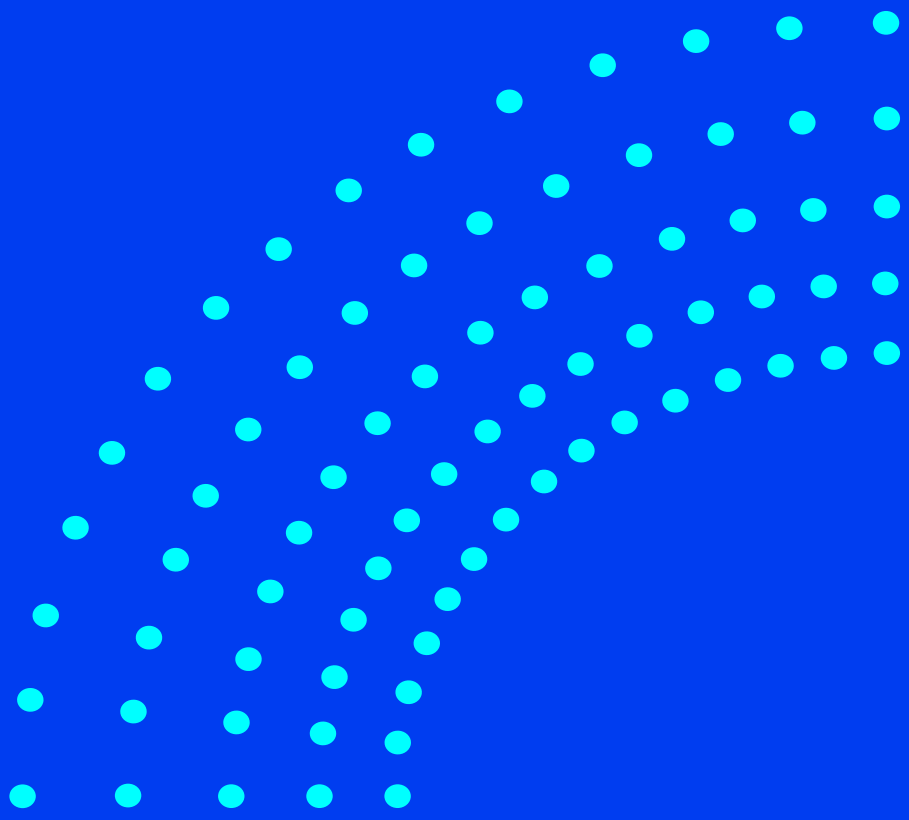
Spelutveckling för handdatorer

Anders Rahm-Nilzon

Göteborg, Sweden 2004

IT University of Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET



REPORT NO. 2004:47

Spelutveckling för handdatorer

Utvärdering av spelutveckling till mobila plattformar

Anders Rahm-Nilzon

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2004

Spelutveckling för handdatorer
Utvärdering av spelutveckling till mobila plattformar

Anders Rahm-Nilzon

Report no 2004:47
ISSN: 1651-4769
Department of Applied Information Technology
IT University of Göteborg
Göteborg University and Chalmers University of Technology
P O Box 8718
SE – 402 75 Göteborg
Sweden
Telephone + 46 (0)31-772 4895

Chalmers repro
Göteborg, Sweden 2004

Spelutveckling för handdatorer

Utvärdering av spelutveckling till mobila plattformar

Anders Rahm-Nilzon

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

SUMMARY

This master thesis aims to provide an understanding of the problems around game development for mobile devices and all the platforms that exists on mobile devices. To illustrate these problems I have ported a graphical library from Pocket PC to Palm OS. This master thesis describes the design, implementation and evaluation for this graphical platform. My main issue was to examine if it was possible to create mobile game with high graphical performance and make them run on an arbitrary platform.

To evaluate my work I have made a benchmark to see if the graphical platform was equally fast on both Pocket PC and Palm OS. I also took the well known computer game XKobo and ported it from Pocket PC to Palm OS to see if my graphical platform was adequate fast to develop mobile games. The problems encountered during this process are further discussed and results evaluated according to performance and the possibility of producing portable games for mobile devices.

Mobile games have become a growing market and many of the large game development companies have open there eyes for this new market. Nokia, Sony and Nintendo are some of the companies that have realized the potential of mobile games. There are many different vendors on the market and all of them have there own platform and there own hardware. To make game development profitable the major vendors have to unite around a standard or the developers have to create platform independent games. This thesis will discuss if it is possible to create platform independent games with a common code base.

The report is written in Swedish.

Keywords: Mobile games, platform independence, graphical library, mobile platforms.

Innehållsförteckning

Inledning

I dagens samhälle använder fler och fler människor sina handdatorer och mobiltelefoner, inte bara till att ringa med eller att koordinera sina möten utan att skicka bilder, skicka e-mail, spela nätverksspel o.s.v. Dagens tekniker så som Bluetooth, GPRS, WLAN och 3G gör det möjligt att koppla ihop och kommunicera med alla möjliga tänkbara enheter. Jonathan P. Allan [1] menar att dagens handdatorer är underskattade som hjälpmedel till dagens stationära datorer och tror därför att handdatormarknaden kommer att öka. I näringslivet försöker företag att få gemensamma system internt för att kunna kommunicera med sina servrar, skrivare, PC:s, handdatorer och telefoner. Även spelindustrin försöker att koppla samman spelar att möta varandra över olika LAN eller över Internet. Spelindustrin växer och är idag en miljard industri som idag omsätter mer än den gigantiska filmindustrin. De stora speltitlarna har liknande budgetar som de stora filmtitlarna som släpps i Hollywood. Tack vare att Microsoft har så stor andel av marknaden för PC:s och har kunnat sätta standarden för applikationer och plattformar så har PC världen idag ett stort försprång när det gäller utvecklingen gentemot handdatorer och mobila enheter. De stora operatörerna på marknaden har nu börjat insett att spel till mobila enheter kan bli riktigt stort och under Cebitmässan 2004 uttalade sig Nokias speldirektör Pasi Pölönen, ”Vi kan inte längre förlita oss på våra produkter. Vi måste fylla dem med något också och det är den stora skillnaden jämfört med förra året.” Spelindustrin håller på att ta till sig mobilitet och det ser man inte minst på att Sony och Nintendo håller på att ta sig in på marknaden, säger Pasi Pölönen [2].

Den mobila marknaden är i dag uppdelad av ett antal olika operatörer och tillverkare som alla har sin egen standard. Detta medför problem när det gäller utveckling av applikationer till mobila enheter. I PC världen räcker det med att utveckla till en, max två plattformar och man har täckt in i princip hela marknaden men till mobila enheter blir det betydligt fler plattformar att anpassa sig efter. För att lösa dessa problem behöver antingen operatörerna och tillverkarna anpassa sina plattformar till en gemensam standard eller att utvecklarna bygger plattformsoberoende applikationer med en i stort sett gemensam kodbas.

Mobila enheter och plattformar

Idag finns det en mängd med olika aktörer på marknaden när det gäller att ta fram mobila enheter, både telefontillverkare så som Sony-Ericsson, Nokia, Motorola, Siemens och handdatortillverkare så som Toshiba, Palm, HP, Sony och Sharp. Alla dessa aktörer följer sina egna standarder och har sina egna plattformar eller operativ system t.ex. så kör Microsoft sitt SmartPhone OS och Sony-Ericsson och Nokia har Symbian OS, naturligtvis använder de sig inte av samma version, och Palm har sitt eget OS. Marknadsandelarna är ganska jämt fördelade över de stora aktörerna, d.v.s. att det inte finns någon av aktörerna som har monopol som t.ex. i PC-världen där Microsoft är den stora aktören och kan sätta sin egen standard. I Europa, sista kvartalet av 2003, såg marknaden ut enligt följande när det gäller antalet sålda handdatorer, enligt IDC Press Center [3].

- **HP hade 37 % av marknaden**
- **Palm hade 30 % av marknaden**
- **Medion hade 10 % av marknaden**
- **Sony hade 8 % av marknaden**
- **Dell hade 4 % av marknaden**
- **Övriga aktörer hade 10 % av marknaden**

För telefoner såg marknaden ut så här, sista kvartalet 2003 för hela världen, enligt IDC Press Center.

- **Nokia hade 33 % av marknaden**
- **Motorola hade 13 % av marknaden**
- **Samsung hade 10 % av marknaden**
- **Siemens hade 9 % av marknaden**
- **LG Electronics hade 5 % av marknaden**
- **Övriga aktörer hade 30 % av marknaden**

Spel i ett lärande perspektiv

Spel används idag av alla olika typer av människor över alla åldrar. Det kan vara när man väntar på bussen för att ta sig hem som tidsfördriv, eller spela olika sport spel mot kompisar för att se vem som är bäst, eller vara ett yrke där man åker runt i världen för att tävla mot andra och vinna prispengar för att ha råd att leva och ta sig till nästa tävling. Det finns massor av användningsområden för spel, de kan fungera som tidsfördriv, nöje och samtidigt vara väldigt lärorika. I artikeln ”Eleven, Leken och Spelet”[4] tar Jonas Linderoth upp sina funderingar samt förankrar dem i vidare forskning. Han tar upp de olika genrer som finns inom dataspel och förklara vad man kan lära sig inom vilken genre. Vidare i artikeln menar han att spel inte kan ses som en enhetlig aktivitet utan när man spelar spel så blandas olika element på olika sätt. Ibland är slumpen av stor vikt för om du ska lyckas eller inte och ibland är det konkurrens som utgör det största hotet till att inte lyckas. Vissa spel utgår från att man själv är i centrum och den verklighet man befinner sig i byggs upp runt sin spelkaraktär beroende på vad man gör för val inom den uppbyggda världen. Jonas avslutar artikeln med vad vi kan lära oss av spel t.ex. kan man med ett simuleringsspel lära sig hur man kör en viss typ av fordon, man kan lära sig historia i en virtuell värld där man själv kan vandra om kring bland dinosaurier och andra djur som inte finns i våran värld, eller utforska rymden genom att själv få åka runt och se olika galaxer och planeter. Dessutom belyser han motivationsfaktorn som spel kan ge till att vilja lära sig nya saker. Jonas får stöd för sina idéer av många forskare världen över bl.a. av Kurt Squire i hans artikel ”Cultural Framing of Computer/Video Games” [5] där han har undersökt människors sätt att se på dataspel och även hur de används och kan användas. Kurt menar att dataspel kan vara så väl socialt som lärande och tar upp *Civilization III* där spelaren måste lära sig att analysera terrängen för att kunna hitta det bästa stället för att sätta upp en stad, eller spel som *Half Life* där militär kan öva sig i att utföra vissa stridsövningar och där de kan öva tillsammans med hela gruppen.

Som jag visade i föregående stycke där jag tar upp marknaden och hur stora andelar de stora aktörerna har på marknaden så kan man enkelt se att utvecklare av spelapplikationer till mobila enheter inte har så lätt om de vill nå ut till större delen av marknaden. När marknaden ser ut så här och de stora aktörerna vägrar att enas om en gemensam plattform eller ett gemensamt operativsystem så blir plattformsoberoende något väldigt viktigt. Ett plattformsoberoende spel kan säljas till alla på marknaden medan man begränsar sig till vissa om man gör ett spel för en plattform. Dessutom riskerar man att den plattform som man själv utvecklar till kanske dör ut eller köps upp av en annan aktör och då måste man börja om från början med sitt utvecklande

Syfte och frågeställning

Syftet med denna rapport är att undersöka om det är möjligt att utveckla plattformsoberoende spelapplikationer med en gemensam kodbas för mobila enheter med olika operativsystem. För att undersöka detta skall ett grafikbibliotek portas från Pocket PC till Palm OS och en spelapplikation skall portas från Pocket PC till Palm OS, med hjälp av detta grafikbibliotek, som skall gå att använda på bägge dessa operativsystem.

Uppsatsen ska svara på följande frågor:

- *Går det att utveckla spel med en gemensam kodbas för mobila handhållna enheter som använder olika operativsystem?*
- *Om ja, krävs det några specifika verktyg och plattformar för att genomföra detta?*
- *Om ja, hur bör en sådan applikation designas och implementeras?*

Terminologi

API	Application Programmer Interface, uppsättning funktioner och parametrar för en programmerare som använder sig av till exempel ett bibliotek eller ett operativsystem.
ARM	Advanced RISC Machines, mikroprocessor teknik utvecklad för att hålla låg kostnad, ha en hög prestanda och vara energisnål.
Bakåtkompatibelt	Att kunna köra samma program på äldre enheter än den som programmet utvecklades för.
Big Endian	Detta gör att en variabel som är två byte stor skiftas och sätter den första byten sist ex. Little Endian: 1110 1111 0111 1011 Big Endian: 0111 1011 1110 1111
Blit	Ritar upp något på skärmen eller på en ritbar yta
Communities	Samhällen eller egna grupper med människor som har liknande intressen.
Event	En händelse till exempel en knapptryckning eller en kollision.
Freeware	Program som är framtagna för att vara gratis att ladda hem från Internet.
Grafikbibliotek	En uppsättning funktioner som förenklar för utvecklaren att få tillgång till skärmen och att kunna rita olika figurer så som linjer och rektanglar.
GNU	GNU Not Unix, är ett projekt som har skapat för att producera ett öppet operativsystem. GNU är ett operativsystem som är väldigt likt UNIX men som är gratis för alla att ladda hem och att ändra i hur de vill, men om man gör en ändring får man inte ta betalt för den.
Handshake	En slags överenskommelse mellan två datorer.
Little Endian	Little Endian är motsatsen till Big Endian, se Big Endian. Little Endian är det vanligaste när det gäller variabler.
Motorola 68k	En äldre processor som gjordes av Motorola och var väldigt vanlig förr i tiden.
Pixel	En pixel är den minsta del av en skärmyta. Skärmens storlek anges ofta i antal pixlar.
Porta	Överföra ett program och dess funktionalitet till en annan plattform.
Sprite	En sprite är en liten bild till exempel en motståndare i ett spel eller ditt eget skepp.
SDK	Software Development Kit, en uppsättning program och verktyg för att utveckla programvara till exempelvis ett annat program eller annat operativsystem.
VRAM	Video RAM, ett video minne för minnet till skärmen. Vissa enheter har ett separat VRAM andra har allt minne i det dynamiska minnet.
Växlar	Växlar till UNIX kommando är olika val man kan göra beroende på vad man vill använda den specifika funktionen till. Till exempel gcc –o så kompileras objektkoden.

WAP	Wireless Application Protocol, WAP är ett protokoll för att kunna visa Internet sidor på bärbara enheter till exempel mobiltelefoner eller handdatorer.

Relaterat Arbete

Detta kapitel innehåller en del av den forskning, inom det område som är intressant för rapporten, för att ge en bild av vad som har gjorts tidigare inom detta område. Kapitlet kommer även att presentera en del applikationer som har gjorts och dess plattformar.

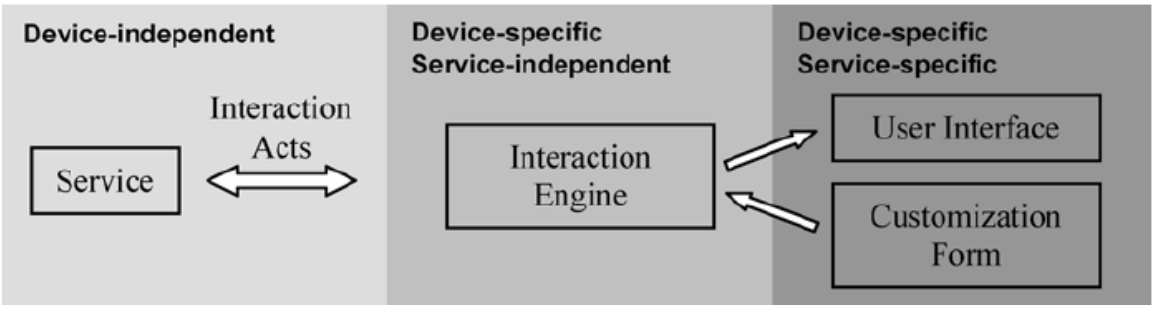
För att ta reda på vad som hade gjorts hittills och var marknaden var på väg läste jag forskningsartiklar, publicerade rapporter från olika företag och statistiksidor på Internet för att bilda mig en uppfattning om vad som kan tänkas hända inom en snar framtid och dessutom vad som har gjorts fram tills idag.

Forskning

I början när de första handdatorerna släpptes så blev det en stor hysteri kring dessa mobila enheter som skulle konkurrera ut de stationära datorerna. Enligt många har just detta synsätt på handdatorer bidragit till att handdatorn som enskild pryl har kallats den mest överskattade IT prylen genom alla tider. Enligt Jonathan P. Allan [1] i en artikel om vad han anser har gått snett i handdatorutvecklingen är inte handdatorn så fruktansvärt överskattad som analytikerna har förutspått under de åren handdatorerna har funnits. Jonathan hävdar att det finns tre faser i handdatorutvecklingen, den första var när handdatorerna uppfattades som pennbaserade datorer, den andra som PDA(Personal Digital Assistant) och den tredje som en ”bärbar kompanjon” (Handheld Companion). Jonathan håller givetvis med om att när de första handdatorerna kom ut på marknaden och lanserades som pennbaserade datorer så var förväntningarna skyhöga och alldeles tagna ur luften. Marknaden var inte mogen för dessa enheter och det fanns inget behov av dem. Därför blev förväntningarna alldeles för högt ställda och därför så slog inte handdatorerna i den utsträckning som var förutspått. Efterhand utvecklades tekniken och tanken att handdatorerna skulle ersätta stationära PC:s försvann och istället ersattes av att handdatorer skulle bli komplement till de stationära datorerna. PDA gjorde sitt inträde på mobilmarknaden. Då menar Jonathan att analytikerna hade blivit lite mer försiktiga i sina förväntningar och tonade ner dem lite, men hade fortfarande väldigt högt ställda förväntningar när det gällde försäljning och spridning av handdatorer. PDA marknaden nådde inte riktigt upp till dessa högt ställda förväntningar, men var inte långt ifrån. Den tredje och sista fasen, enligt Jonathan är den som vi ser idag. Den Jonathan kallar för ”bärbara kompanjoner”. Nu anser Jonathan att analytikerna är väldigt försiktiga och rent av underskattar marknaden för framtiden. Enligt Jonathan kommer marknaden för handdatorer att öka mer än vad analytikerna förutspått de närmaste åren.

För att belysa problematiken med plattformsoberoende applikationer har jag valt att titta lite närmare på två forskningsprojekt där båda har som mål att sätta samman olika system men har lite olika infallsvinklar vad det gäller plattformsoberoende. I artikeln ”Mobile access to Real-Time Information – the case of autonomous stock brokering” [6], berättar Stina Nylander et. al. om svårigheterna att överföra applikationer till mobile enheter. De hävdar att de applikationerna som finns idag för handdatorer som har tillgång till realtids information har en restriktiv funktionalitet och väldigt liten har gjorts för att anpassa

applikationen till de mobila enheterna, så som begränsat dynamiskt minne och mindre displayer. De menar också att den nuvarande metoden för att skapa applikationer till olika enheter är att göra olika versioner och tar upp WAP som exempel. Idag finns mängder av olika mobila enheter ute på marknaden och givetvis blir det ett problem om man ska utveckla en version av sin applikation för varje enhet. Deras lösning är att utforma funktionaliteten så att den från början är anpassad för multipla plattformar och dessutom få applikationen att automatiskt anpassa sig till den enheten som den körs på. De har testat sina idéer genom att utveckla en tjänst som de kallar för TapBroker, som är en tjänst som ska ge information och stöd för användaren om börsen. TapBroker är utvecklat med JAVA och bygger på PUSH teknologin. Dessutom har de använt sig av Ubiquitous Interactor (UBI) som ska göra så att applikationen ska gå att köra på vilken enhet som helst. UBI fungerar så att utvecklaren utvecklar en tjänst och sen specificerar hur denna tjänst ska visas eller se ut på de olika enheterna. Information om nya enheter kan läggas till när som helst.



Figur 9. Visar UBIs arkitektur och hur det är uppbyggt. Dessutom visas vilka av de olika stegen som är plattformsoberoende.

I artikeln ”The Children´s Machines: Handheld and Wearable Computers Too” talar Bakhtiar Mikhak et. al. [7] om barn som *computer generation* och menar att barnen i vårat samhälle kommer att ärva de datorsystem som vi gör idag och att det kommer att vara dem som kommer att utveckla datorsamhället på grund av att de är uppväxta med datorerna och kommer således att ta dessa förgivet. De belyser även vikten av att få datorer och olika typer av enheter att fungera tillsammans så att människan kan konstruera och designa sitt eget datorsystem för att täcka sina behov. I deras artikel har de tagit fram sensorer som de kallar för *Crickets* och programmerbara klossar som de sedan har låtit barn ”leka med” och komma fram till designförslag genom att använda sensorerna och de programmerbara klossarna till att göra sin vardag roligare. Barnen kom fram till många olika områden som de kunde tänka sig att dessa sensorer och klossar kunde vara skoj att ha, t.ex. så gjorde ett barn en hastighetsmätare till sina inlines där hon satte fast en magnetsensor på sin inlineskridsko och fäste en magnet i ena hjulet och varje gång den passerade sensorn så genererade sensorn en signal. Hon konstruerade även en tidtagning så att hon kunde ta tiden mellan signalerna och kunde på så sätt få fram en hastighet. Ett annat exempel från artikeln är *Musical friendship Ring* som togs fram av fem kompisar som använde sensorerna till att göra ett musikstycke. De utgick från ett musikstycke med fem instrument som de laddade varje sensor med, varje sensor kunde bara spela ett instrument vardera. Sensorn utrustades

med en knapp och tryckte man på den knappen spelade enheten upp sitt instrument. De programmerade varje sensor till att skicka ut ett unikt ID för att identifiera sig själv till de andra sensorerna. När man sedan förde dessa fem Crickets tillsammans så gjorde de *Handshake* och synkroniserade sig och sen spelade de upp musikstycket med alla fem instrumenten.

Som det framgår av dessa två artiklar så ställer det höga krav på applikationerna och dessutom på designen när man ska konstruera plattformsoberoende applikationer. I den första artikeln används en plattform som tar hänsyn till olika enheter och dess egenskaper och i den andra artikeln används sensorer som ges olika egenskaper beroende på vilka system som ska kopplas ihop och vilka egenskaper som ska framhävas.

Även Johan Sanneblad och Lars Erik Holmquist tar upp vikten av att ha en bra plattform, när man programmerar spel till mobila enheter, i sin artikel ”Prototyping Mobile Games” [8]. De skriver att skillnaden mellan att programmera spel till stationära datorer och till mobila enheter är att till stationära datorer finns DirectX [9], som gör det möjligt att få tillgång till hårdvarufunktioner på stationära datorer. Dessutom underlättar DirectX när man ska programmera grafik till applikationer för den stationära datorn. Till mobila enheter finns inte DirectX utan där får programmeraren göra allt själv eller försöka hitta något grafikbibliotek, se rubriken ”Plattformar och grafikbibliotek”, som passar för just den enheten som man vill utveckla till. Johan och Lars Erik har därför utvecklat Open Trek, som är ett forskningsprojekt under Viktoria Institutet, som ska ge utvecklaren det stöd till mobila enheter som DirectX ger till stationära datorer. Författarna menar att utan en bra plattform tar det alldeles för lång tid att utveckla grafikapplikationer så som spel till mobila enheter. Open Trek är uppbyggt av två delar, den ena är GapiDraw, som jag tar upp mera under rubriken ”Plattformar och grafikbibliotek”, och är ett grafikbibliotek som underlättar själva grafikprogrammerandet och utformningen av applikationen och den andra är en nätverksdel som underlättar utvecklandet av nätverksapplikationer för mobila enheter. Att kunna utveckla applikationer med nätverksstöd gör att det blir möjligt att utveckla multiplayer spel d.v.s. flera spelare kan spela samtidigt. Författarna testade sin plattform i en klass som läste sin magisterexamen i mobil informatik och samtliga grupper kunde utveckla ett spel på projekttiden fem veckor. Alla spelen var multiplayer spel och två av spelen hade till och med stöd för mer än två spelare. Totalt konstruerades 12 spel och grupperna var indelade i 1, 2 eller 3 personer. När det gäller ämnet spel till mobila enheter så kommer det att komma fler spel som kan spelas med flera spelare samtidigt. Denna teori stöds av bland annat Dan Bendas och Mauri Myllyaho i deras artikel om trådlösa spel [10]. De menare att de absolut vanligaste underhållningsapplikationerna idag för mobila enheter är spel som är inbyggda i enheten. Detta tack vare att en sådan applikation slipper att ta med en nätverkslösning som kan göra så att applikationen går mycket saktare, men även att det beror på att operatörerna inte har samarbetat med utvecklare för att få fram en bra nätverksplattform som inte kostar så mycket att använda och som är funktionell och pålitlig. Författarna tror att det kommer att ske en stor utveckling på multiplayer spel för mobila enheter, men att då måste det komma en standard som både operatörer och utvecklare kan godta. Dan och Mauri menar att multiplayer spel kommer att bli populära tack vare att det gynnar både användaren, då det ger en större utmaning att spela mot en

annan människa och operatören därför att multiplayer spel hjälper till att bilda *communities* av lojala kunder. De har utvecklat ett klassiskt kortspel som heter ”Four Players’ Tuppi”, som har en liknande spelidé som Bridge eller Whist, för att visa sina teorier. Spelet tror de har stora möjligheter till att lyckas då de har:

- *En oberoende nätverksdel*, det räcker att använda sig av någon av de olika teknikerna som finns idag t.ex. IR eller Bluetooth bara tekniken kan skicka och ta emot data.
- *Oberoende av vilken enhet som används,* så länge som enheten har någon form av programmeringsspråk eller att man kan implementera applikationer på den.
- *Lätt att porta till en annan plattform,* deras implementation stöder sex olika plattformar och modellen är väldefinierad.

På Viktoria Institutet i Göteborg bedrivs forskning inom IT området och Alexandra Weilenmann har deltagit i ett projekt där de tillverkade en mobil enhet med en nätverksapplikation som skulle tala om för användare om någon kompis var i närheten. Projektet kallades ”HumingBird” för att den mobila enheten hade en vibratorfunktion som satte igång om någon av ens kompisar var i närheten och enheten brummade som en humla. För att utveckla enheten använde de sig av Nintendos GameBoy och även Nintendos plattform. [11]

Figur 10. En Humming Bird.

Ett närliggande arbete till det jag har gjort är Jonas Bylund och Erik Ruisniemis arbete, GapiDraw för Symbian [12] som tar upp problematik och möjligheter för spelutveckling till mobiltelefoner. Jonas och Erik har kommit fram till att det är svårt och mycket att tänka på när man ska skriva plattformsoberoende kod. De kom fram till att det inte gick att lyfta över ett spel skrivet för Win32 eller SmartPhone direkt till Symbian OS med hjälp av GapiDraw utan en del modifikationer. Det som skiljer mitt arbete från deras är att jag koncentrerar mig på en plattform som är skriven för handdatorer och handdatorer är mer lika stationära datorer än vad mobiltelefoner är idag. Jonas och Erik har varit till stor hjälp för mig under utvecklingsarbetet och de har delat med sig av deras erfarenheter.

Spelplattformar och grafikbibliotek

För att utveckla grafiska applikationer idag så underlättar det om man har en grafikplattform eller ett grafikbibliotek. Idag finns det ett antal olika grafikbibliotek ute på marknaden men de mest använda är GapiDraw, MobileCore, Mophun och PocketFrog.

Gapi

Gapi(Game Application Programming Interface) är en samling funktioner från Microsoft som gör att man får en adress till skärmytan på Pocket PC eller SmartPhone. Det finns också funktioner för färgdjup, bredd och höjd på skärmen och man kan även låsa knapparna på enheterna. Dessutom kan man ta reda på hur många steg i minnet man behöver gå för att få nästa pixel i x-led och i y-led. Detta ger programmeraren en ren yta att konstruera sitt spel på, vilket underlättar då programmeraren inte behöver lägga ner tid på att skapa fönster och knappar o.s.v. Gapi är däremot inte något grafikbibliotek utan är en samling funktioner som hjälper programmeraren att ha kontroll på hårdvaran.[12]

GapiDraw

GapiDraw är ett grafikbibliotek för utveckling av grafiska applikationer till mobila enheter. GapiDraw har stöd för Microsoft Windows, Microsoft Pocket PC, Microsoft SmartPhone och Symbian OS. GapiDraw släpptes den 22 mars 2002, men hade testats tidigare under en kurs på IT-Universitetet om mobil spelutveckling. Dåvarande namnet var GapiTool och byggde på Microsofts spel API GAPI, men efter en hel del optimeringar så bytte GapiTool namn till GapiDraw. Efter att GapiDraw 1.0 hade släppts fortsatte arbetet med att optimera koden och det lades även till stöd för olika skärmupplösningar på PC versionen och stöd för olika skärmvarianter. Dessutom optimerades grafikrutinerna genom att lägga in dem i funktions- och klassmallar och funktionerna anpassades till att bli användarvänliga för utvecklarna. Efter hand tillkom fler funktioner och den 8 Maj 2003 hade det tillkommit så många funktioner och optimeringar att det var dags att släppa GapiDraw 2.0. GapiDraw har används flitigt och över 140 olika speltitlar har utvecklats med GapiDraw och den 26 januari släpptes GapiDraw 3.0 nu med stöd för Symbian OS. [13]

MobileCore

MobileCore är ett nytt grafikbibliotek som bara har släppts i en alpha version än. MobileCore ska stödja Pocket PC och Symbian och är ett GNU projekt vilket innebär att det är fritt att ladda hem och utvecklaren själv får inte ta betalt för det som han skapar med MobileCore. Dessutom är hela källkoden fri att ladda hem från nätet. [14]

Mophun

Mophun är en utvecklingsplattform framtagen av fem programmerare som startade företaget Synergenix. Mophun använder sig av en virtuell maskin som kallas *Mophun RTE(Run Time Engine)* som översätter koden till bytekod så att den kan köras på samtliga

enheter. De enheter som stöds är de telefoner som använder sig an Symbian version 6.1 eller högre samt några Sony Ericsson modeller. Mophun har ett eget API som är gjort i C-kod och har de vanligaste funktionerna när det gäller 2D-grafik så som blitta till skärm, rita rektanglar, sätta ut pixlar och rita linjer, men mophun har även stöd för ljud, trådhantering, sms och även 3D-grafik. För att få ladda hem Mophun SDK måsta man registrera sig på Mophuns hemsida och för att få distribuera Mophun GameLets som Mophun kallar de applikationer som skapas med *Mophun API* så måste detta ske via Synergenix hemsida eller via ett företag som har *Mophun VST(Mophun Vendor Signing Tool)* och rättigheter att distribuera gamelets från Mophun. Alla gamelets blir låsta till en enhet.

Mophuns olika paket:

Mophun RTE (Run Time Engine). Detta är den virtuella maskinen.

Mophun API (Application Program Interface). Ett antal funktioner för att använda RTE.

Mophun SDK (Software Development Kit). Funktioner för att kontrollera API:et.

Mophun VST (Vendor Signing Tool). Gör så att en gamelet går att köra på en vald enhet och låsa gamelet:en till den enheten. [15]

PocketFrog

PocketFrog är ett grafikbibliotek liknande GapiDraw som också det är baserat på GAPI. PocketFrog erbjuder en rad enkla funktioner så som rita linjer och rita rektanglar men erbjuder inte ljudbibliotek eller hantering av sprites. PocketFrog är skrivet i C++ av Thierry Tremblay som en freeware. Till skillnad från GapiDraw stödjer PocketFrog bara Pocket PC och inte stationära plattformar eller SmartPhone. Thierry Tremblay har också skapat PocketHAL som släpptes den 14 mars 2004. PocketHAL är på samma sätt som GAPI ett bibliotek för att kontrollera hårdvaran på enheterna och PocketHAL ska vara effektivare och snabbare än GAPI. [16]

Nätverksapplikationer

Internet Processor Chip

Företaget EG Components har utvecklat ett Internet Processor Chip (iPC) som ska göra det möjligt att få alla möjliga enheter att kommunicera med varandra via Internet som t.ex. Kylskåp, tvättmaskiner, luftkonditionering, kameror etc. iPC är ett chip utrustat med en ARM processor, ett flash minne, SDRAM och ett litet realtidsoperativsystem vid namn VxWorks. iPC är testat på en liten robot och en karta av lego, där roboten kunde navigera sig fram med hjälp av färgerna på lego.[17]

Näringslivet

”I den pågående övergången från industri- till informationssamhälle är effektiv hantering av data och information avgörande för konkurrenskraften. Näringslivet har insett att nätverksapplikationer förbättrar kommunikationen, såväl inom som mellan företag och deras partners, underleverantörer och kunder.”, så skrev VD:n för utvecklingsföretaget Nocom, Anders Jonson, 1999 i årsredovisningen för 1998[18]. År 2004 har förtaget blivit ett av Sveriges ledande IT-företag med start position inom flera branscher så som industri, handel, media, telekom, offentlig sektor samt bank och finans.

Spel

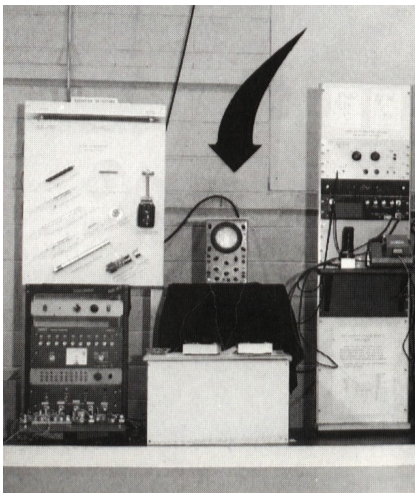
De absolut mest spelade titlarna när det gäller PC-spel är de spel som har nätverksstöd och stöd för flera spelare. Speltestningssiten 1-UP, har listat top 5 bland PC-spel och samliga har nätverksstöd för multiplayer [19].

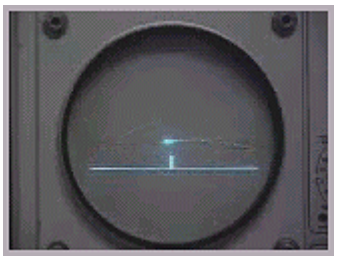
Spelhistorik

Detta kapitel kommer att innefatta spelhistorik från att det första dataspelet, som uppkom på ett oscilloskop vid ett museum i USA, till de stora speltitlarna som idag omsätter miljoner och har en budget större än de flesta hollywoodfilmerna

Historia

1889 startade Fusajiro Yamauchi ett företag som gjorde spelkort. Dessa kort exporterades till västvärlden 1907, men det är inte detta som är intressant när det gäller detta företag utan det som gjorde det intressant för oss i dataspelsvärlden var att detta företag ändrade namn till Nintendo Playing Card Co Ltd 1951 och var grunden till de gigantiska spelföretag som Nintendo är idag, med speltitlar som Super Mario Bros och Zelda [20]. Men det är längre fram i historien och det hände mycket innan dess. 1891 startar Gerard Philips ett företag i Holland som specialiserar sig på lampor och andra elektroniska produkter. Företaget heter idag Phillips Co och är en av de största aktörerna när det gäller elektroniska komponenter. 1918 skapar Konosuke Matsushita, Matsushita Electric Housewares Manufacturing Works som under de närmaste 70 åren startar upp en mängd olika företag bl.a. Panasonic Co. När vi kommer till 1947 har två japaner vid namn Akio Morita och Masaru Ibuka blivit väldigt intresserade av en amerikansk pryl som vi idag kallar för en bandspelare. Dessa båda herrar tar med sig radion hem till Japan och konstruerat om bandspelaren till att bli världens första fickradio och den blir en stor succé i Japan. När sen Japanerna vill marknadsföra dess i USA och Europa så har engelsmännen svårt med företagsnamnet så japanerna döper om sig till Sony. En ny aktör är född som har stor inverkan på dataspelshistorien. 1951 få Ralph Baer i uppdrag att konstruera den bästa TV som någonsin byggts, Ralph vill att Tv:n ska innehålla interaktiva spel men nekas av ledningen. 1954 skapar David Rosen spelföretaget SEGA som står för ”SErvice GAmes”. 1958 skapade Willy Higinbotham ett spel som han kallade för PONG – Tennis för två, se Figur 1 och 2.





Figur 1-2. Oscilloskopet för Tennis för två till vänster och själva skärmen till höger.

Higinbotham tyckte att besökarna till Brookhaven National Laboratories såg uttråkade ut och skulle med hjälpa av detta spel försöka att förhindra detta. 1961 skapade en student vid namn Steve Russel på MIT det första interaktiva dataspelet på en Digital PDP-1 och använde sig av den nya CRT-skärmen för att visa grafik. Spelet heter SpaceWars och går ut på att skjuta ner ett annat rymdskepp.

Figur 3. SpaceWars

Under 60-talet fortsätter Baer med sina idéer om att bygga in spel i TV och lyckas tillräckligt bra för att ett företag som heter Magnavox köper hans idéer, men 1971 kommer ett annat företag, Nutting, ut med det första arkadspelet med en 13 tums svart/vit TV inbyggd och med handkontroller som de kallar för Computer Space, Figur 4, men allmänheten tyckte att det var alldeles för svårt att spela spelen så det blev ingen storsäljare. 1972 lanserar Magnavox sin arkadmaskin där det ingår ett antal spel bl.a. video tennis. Nolan Bushnell and Ted Dabney startar Atari och bygger en arkadmaskin med spelet PONG, spelet blir en stor succé. Strax efteråt släpper Magnavox världens första spelkonsol och den säljer bra för det är det närmaste allmänheten kan komma en hemmaversion av PONG.

Figur 4. Computer Space

Under resten av 70-talet dominerar Atari marknaden och släpper en del konsoler och en del nya spel bl.a. ett spel som heter Death Race som efter att ha ansetts för våldsamt dras tillbaka från marknaden. 1978 ger sig Nintendo in på marknaden med en version av spelet Othello, Atari släpper ett spel som heter Football och Taito släpper spelet Space Invaders. Atari har fått konkurrens men Football går väldigt bra och Space Invaders tillsammans med Football slår alla tidigare försäljningsrekord. 1980 släpper Namco Pac-Man världens populäraste arkadspel. Än så länge har Nintendo haft det tungt och deras titlar har inte haft någon genomslagskraft på marknaden, men 1981 släpper man spelet Donkey Kong, i den så kallade ”Game&Watch”-serien som är bärbara spelenheter med ett spel i, med hjälten Jumpman som senare döps om till Mario vilket kommer visa sig bli en väldigt framgångsrik karaktär. Samtidigt når USA en notering på 5 miljarder dollar på inspelade pengar på arkadspel och amerikanarna har lagt ner 75000 timmar på att spela dataspel. 1985 testar Nintendo sitt NES (Nintendo Entertainment System) och samma år skapar ryssen Alex Pajitnov, Tetris ett beroendeframkallande pusselspel som kan spelas på PC som senare kommer visa sig bli det spel som har sålt bäst av alla titlar fram tills idag. 1986 släpper Nintendo sitt NES efter att ha gått bra på testmarknaden New York, dessutom släpper Nintendo Super Mario Bros som blir en super succé. SEGA släpper sin första konsol, men den blir snabbt utkonkurrerad av NES. 1987 släpper Nintendo Kid Icarus och Metroid och får snabbt ett monopol över konsolmarknaden. 1989 är ett stort år för de mobila enheterna då Nintendo släpper sitt GameBoy som är den första portabla spelstationen i världen. Trots en väldigt liten svart/vit skärm så blir den en jättesuccé, Figur 5 visar ett GameBoy.

Figur 5, GameBoy from Nintendo

SEGA släpper sin spelkonsol SEGA MEGA Drive som har en 16-bit processor. Det stora året för Nintendo blir 1990 med Super Mario Bros 3 som blir det mest sålda tvspels-cartridge genom tiderna och samtidigt lanseras Nintendos 16-bitars konsol i Japan med Super Mario Bros 4 (Super Mario World). Samma år släpper NEC en handburen variant av TurboGrafx-16 och det är första gången som man kan spela arkadspel i en handdator. 1991 släpps SNES (Super Nintendo Entertainment System) i USA som är Nintendos 16-bitars spelkonsol. Dessutom kommer spelet StreetFighter II som blir en stor succé och

arkadspelen får ny luft under vingarna. 1994 släpps strategispelet WarCraft från Blizzard Entertainment till PC och 1995 släpps WarCraft II även det till PC, som båda blir stora succéer [21, 22]. Resten av 90-talet tillbringar de stora tillverkarna med att skapa en 32 eller 64-bitars spelkonsol som ska kunna läsa spelen från en CD-rom skiva. Det släpps maskiner som Phillips CD-i, Electronic Arts 3DO, SEGA's Saturn, Atari's Jaguar, Nintendos N64 och kanske den absolut populäraste Sony Playstation. I slutet av 90-talet börjar det ryktas om konsoler med DVD-läsare och 1999 säger Microsoft att de håller på att utveckla sin X-Box och Sony annonserar sin Playstation 2 i september med en 250MHz RISC processor från Toshiba. 1998 släpps PC-spelet Half-Life som år 2000 får en uppdatering som heter Counter-Strike [23, 24]. Spelet är ett spel där man tillsammans i team utför krigsuppdrag och har möjlighet att spela mot andra över Internet. Spelet har fått mängder av priser och utmärkelser. Samtidigt släpps StarCraft och StarCraft - BroodWar som är uppföljare till WarCraft och har nu stöd för att spela mot varandra över Internet [25]. Även StarCraft har fått mängder av priser och i Kina har StarCraft spelandet egna TV-kanaler där det sänds StarCraft matcher dygnet runt. Playstation 2 släpps den 4 Mars 2000 och efter två dagar har den sålt 1 million exemplar, Figur 6.

Figur 6, Japansk version av Playstation 2 med hårddisk.

Även detta år startar SEGA sin Internet service för Dreamcast och Nintendo släpper en applikation som gör det möjligt att skicka e-mail via sitt GameBoy Color. Nintendo släpper också sin Pikachu 2 GS, Figur 7, som kan utbyta data via IR-porten med ett GameBoy Color.

Figur 7, Pokémon Pikachu 2 GS

2001 släpper Microsoft sin X-Box som har en 733MHz Pentium III processor och 250Mb grafikkort och 8Gb hårddisk, dessutom stöd för Internet så att man kan spela nätverksspel, se Figur 8. [26]

Figur 8, Microsofts X-Box med Internetstöd

2002 XKobo portas till Pocket PC av Leif Kornstaedt med hjälp av grafikplattformen GapiDraw och får namnet PocketKobo, Figur 9.

Figur 9. PocketKobo på en Toshiba e740

Metod

I detta kapitel tar jag upp hur jag gick tillväga för att utföra mitt arbete och för att komma fram till svar på den fråga jag vill undersöka. Här tas upp lite om olika metoder kring programvaruutveckling och varför jag valde den metod som jag valde. Mycket av mitt arbete har gått ut på att hitta lösningar på mina problem via de medier som finns d.v.s. artiklar, forum, hemsidor och böcker.

Förstudie

För att få en klar bild av vad som behövdes göras började jag lägga upp en grov strategi för hur arbetet skulle genomföras. Då jag skulle välja grafikbibliotek för uppgiften stod valet mellan Mophun och GapiDraw, men eftersom jag inte kunde få tillgång till kodbasen för Mophun så föll valet ganska naturligt på GapiDraw. De andra biblioteken var antingen för nya och oprövade eller så riktade de sig inte till plattformsoberoende utveckling, vilket är en förutsättning för detta arbete. Vad det gäller plattformen Palm OS så är det en av de största i Europa, den största i USA och samanlagt så är Palm OS störst i världen när det gäller handdatorer, även när det gäller telefoner är Palm OS väldigt stora. Därför föll mitt val på utveckling till plattformen Palm OS.

Litteraturstudie

Efter att jag hade bestämt mig för grafikbibliotek och för plattform behövde jag inhämta en hel del kunskap om dessa, så jag läste dokumentationen för GapiDraw och satte mig in i klasshierarkin för att få en bra bild av vad som är nödvändigt att lyfta över till Palm OS och vad som ska portas först. Jag fick även sätta mig in i olika utvecklingsmiljöer för Palm OS. Litteraturstudien fortsatte i att läsa om andra plattformsoberoende projekt och även i att läsa om grafikprogrammering och spelutveckling i allmänhet. Den största biten var att sätta sig in i uppbyggnaden av Palm OS och försöka skapa sig en uppfattning om vad som gick att porta till Palm OS och hur man skulle gå tillväga. Mycket av kunskapen hämtade jag från Palms egen dokumentation och forumet Palm-dev-forum.

Analys

Analysen av detta arbete gick ut på att få en bild av vad GapiDraw skulle klara av på Palm OS. Till att börja med skulle det vara ett högpresterande grafikbibliotek för de plattformar som stöds d.v.s. så även för Palm OS. Dessutom ska GapiDraws gränssnitt se likadant ut för en utvecklare, så att det med minsta möjliga arbete går att kompilera om koden för en annan plattform. Det ska alltså gå att flytta över sin applikation till en annan plattform genom att bara kompilera om sin kod för den tänkta plattformen.

Design

När det gäller designen så fanns ju redan en design för GapiDraw eftersom att GapiDraw har funnits till Pocket PC och SmartPhone. För att få utvecklarna som använder detta

bibliotek att känna igen sig i nästa version av GapiDraw så var det ett krav att designen skulle vara intakt i möjligaste mån. Om det var tvunget att ändra i designen så skulle det även fungera på existerade plattformar så som Pocket PC och SmartPhone.

Evaluering

För att utvärdera plattformsoberoende applikationsutveckling gjorde jag en benchmark där jag testade ett antal funktioner på Win32, Pocket PC och på Palm. Dessutom lyfte jag över en spelapplikation från Pocket PC till Palm OS för att kunna jämföra de olika plattformarnas prestanda och se om det ens var möjligt att köra en spelapplikation på Palm OS utvecklat med ett plattformsoberoende verktyg. Se vidare under rubriken Resultat för att få utförligare beskrivning om hur testerna gick till och vad jag kom fram till för resultat.

Genomförande

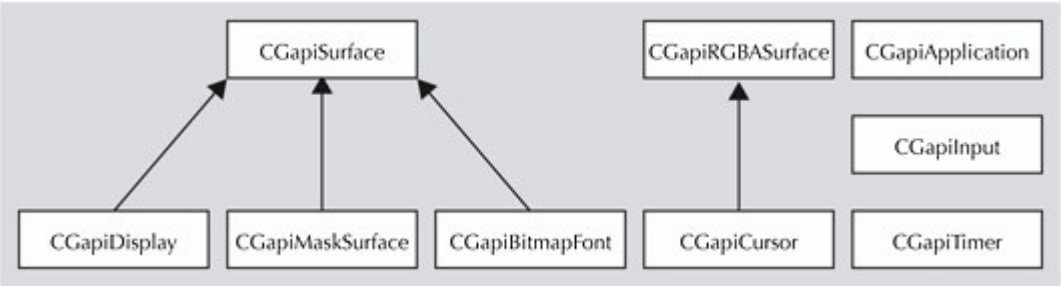
Detta kapitel går igenom hur jag utförde projektet. Det går igenom utvecklingsmiljöer för att utveckla applikationer till Palm OS, GapiDraw, portningen av GapiDraw till Palm OS och även portningen av PocketKobo, ett Pocket PC spel som jag lyfte över till Palm OS för att kunna testa prestandan på GapiDraw för Palm OS.

Projektstart och Utvecklingsmiljöer

För att komma igång med projektet var jag tvungen att sätta mig in i, för det första Palmplattformen och för det andra grafikbiblioteket GapiDraw. De första veckorna gick åt till att studera dessa båda plattformar parallellt. Jag fick även bekanta mig med de utvecklingsmiljöer som fanns för Palm OS och det var GCC(GNU C Compiler) och Metrowerks CodeWarrior. Från början bestämde jag och min handledare oss för att vi skulle köra på GCC som är en textbaserad kompilator och för att ställa in olika funktioner använder man sig av olika växlar t.ex. –o för att skapa objektkod. Detta för att GCC är freeware och för CodeWarrior behövs en licens som kostar pengar. Tanken var att om det gick att porta GapiDraw med GCC så går det även att utveckla applikationer med GCC och då behöver inte utvecklarna skaffa dubbla licenser, en för GapiDraw och en för CodeWarrior, för att utveckla till Palm OS. Tyvärr så blev det alldeles för krångligt att utveckla med GCC då det behövs ställas in en mängd inställningar och växlar för att kunna utveckla applikationer till Palm OS och det skulle ta för lång tid att porta GapiDraw med GCC så det blev CodeWarrior som jag använde som utvecklingsmiljö.

GapiDraw

GapiDraw styrs av klassen CGapiApplication. I CGapiApplication finns initieringen för skärmen och själva spelloopen, som tas upp av de Sousa i Game Programming All In One [27]. Det finns även funktioner som hanterar knapptryckningar och penntryckningar. CGapiApplication är den klass som utvecklaren har tillgång till och även källkoden till och den styr i sin tur en dll(Dynamic Linked Library) som tar hand om de funktionsanrop som görs i CGapiApplication. Figur 10 visar ett klassdiagram för hur GapiDraw är uppbyggt.



Figur 10. GapiDraws klasshierarki.

CGapiDisplay

Denna klass styr skärmhanteringen d.v.s. att det är CGapiDisplay som har kontroll på själva skärmen, den som visas för användaren. CGapiDisplay har även kolla på om enheten har en egen skärmbuffert eller om det behövs skapas en. En skärmbuffert är nödvändig då det går lite för segt att skriva direkt till skärmen och det skulle uppfattas av en användare av applikationen att skärmen flimrar. Att CGapiDisplay är en underklass till CGapiSurface innebär att man kan göra samma operationer på en instans av CGapiDisplay som på en CGapiSurface.

CGapiMaskSurface

CGapiMaskSurface används vid kollisionsdetektering och är också en underklass till CGapiSurface.

CGapiBitmapFont

Med hjälp av denna klass är det möjligt att skapa egna typsnitt. För de som inte känner för att skapa egna typsnitt finns det inbyggt i systemet två typsnitt som går att använda.

CGapiSurface

CGapiSurface är den klass som skapar *ytor* att rita på. Dessa är nödvändiga i GapiDraw för att kunna utföra saker som att infoga bilder eller rita linjer. När man har skapat en yta är det bara att använda de funktioner som finns t.ex. sätta ut pixlar, skapa en bild, rita en linje eller rita en rektangel. Om man vill så har även CGapiSurface stöd för att skapa transparanta objekt, så som en genomskinlig bild eller rektangel, men kan även rotera sina objekt eller zooma dem till en annan storlek. CGapiSurface har även stöd för olika färgformat. De olika format som finns idag är 565(65536 färger), 555(32768 färger) och 444(4096 färger). Siffrorna står för hur många bitar av varje färg som ska representeras och det är RGB kodat vilket betyder att den första siffran står för hur många bitar som ska representera Rött och den andra siffran står för hur många bitar som ska representera Grön och den sista siffran står för hur många bitar som ska representera Blått. För utvecklaren så är detta inget man behöver kunna då man anger en COLORREF som tar ett värde mellan 0-255 för varje färg och sedan konverterar CGapiSurface detta till rätt format beroende på vilket enhet man använder.

CGapiCursor

Denna klass används bara på stationära datorer och gör det möjligt att rita ut muspekaren på skärmen. Klassen är en underklass till CGapiRGBASurface.

CGapiRGBASurface

Detta är en speciell typ av yta som stödjer bilder med en speciell alphakanal. Alphakanalen anger hur genomskinlig varje pixel ska vara i bilden vilket gör att om två bilder med alphakanaler ritas ovanpå varandra så kommer olika delar av bilden att lysa olika mycket. Varken CGapiRGBASurface eller CGapiCursor bedömde jag som nödvändiga att lyfta över till GapiDraw för Palm OS.

CGapiTimer

Med hjälp av denna klass är det möjligt att ha kontroll på systemklockan och på så sätt kontrollera själva spelloopen så att enheten hinner rita upp allt som den ska under varje varv i loopen. CGapiTimer jämför tiden det tog för loopen att bli klar med den avsedda tiden och om det gick snabbare att genomföra loopen än avsett så ser klassen till att systemet väntar tills det är dags att exekvera nästa cykel i loopen. Denna klass finns bara för att det inte ska bli någon skillnad när samma applikationer exekveras på olika snabba processorer. Även denna klass har jag valt att inte implementera i GapiDraw för Palm OS då Palm OS systemklocka inte har den noggrannheten som Pocket PC och det skulle betyda att man inte kan vara säker på att den väntar rätt antal millisekunder.

CGapiInput

CGapiInput är den klass som har kontroll på hårdvaruknapparna. CGapiInput låser de knappar som inte ska ge ett event, T.ex. så vill användaren inte att kalendern ska komma fram om användaren sitter och spelar spel och ska hoppa eller skjuta eller vad som kan tänkas hända i ett spel. Dessutom har klassen kontroll på om skärmen roteras och mappar om upp-, ner-, vänster- och högerknapp till de värden som de får när skärmen roteras 90, 180 eller 270 grader.

Grafik i Palm OS

Palm OS använder sig av något de kallar för Form Manager vilket gör att man kan skapa olika forms, detta är en motsvarighet till fönster på Microsofts plattformar. I Form Manager finns det massor av funktioner så som Open, Create och Close. Om man vill rita olika objekt så använder man sig av Windows Manager som har kontroll på färgen på pennan, grafikstatus och olika mönster. Windows Manager innehåller många olika ritfunktioner så som WinDrawBitmap, WinDrawLine och WinDrawRectangle. Anledningen till att man inte vill använda dessa är att de inte har direktaccess till skärmen och Palm OS i sig bestämmer när det finns tid för att utföra dessa operationer och som jag har varit inne på tidigare så är inte systemklocka tillräckligt precis för att kunna utföra dessa operationer tillräckligt ofta som det behövs i t.ex. ett spel.

Datatyper i GapiDraw

I GapiDraw finns det en rad olika datatyper som inte ingår i standard C++ utan är windowsspecifika. Detta gjorde att jag fick skapa en headerfil där jag definierade de olika datatyperna som ingick i GapiDraw som t.ex. HRESULT, DWORD, RECT, TCHAR med flera. Dessa typer definierades för att utvecklarna ska behöva ändra så lite som möjligt för att flytta över sina applikationer till Palm OS.

Palm OS vs Pocket PC

Efter att ha gått igenom källkoden för GapiDraw och dokumentationen för utveckling till Palm OS kom jag fram till att det var en del som skulle bli svårt att överföra rätt av. För det första har Palm OS har inget eget filsystem och eftersom att GapiDraw länkas via en dll så kommer detta att utgöra ett problem, för det andra så är CGapiApplication skriven för att hantera fönster och trådar vilket är Microsofts sätt att hantera applikationer. När det gällde dll problemet så har Palm OS två olika sätt att hantera detta. Det ena är shared library och det andra är static library. Static library länkas in i koden direkt vid kompileringen medan shared library blir en resurs som flera program kan använda. Shared library är det alternativet som är närmast dll lösningen på Windows, men då uppstår ett problem med alla globala variabler som finns i GapiDraw därför att shared library stödjer inte globala variabler. Det andra problemet med CGapiApplication går inte att komma ifrån utan att skriva om klassen så att den anpassas till Palm OS. Efter att ha överlagt det jag hade kommit fram till om dll:er och CGapiApplication med min handledare så kom vi överens om att båda dessa problem inte ingick i detta arbete utan jag fick koncentrera mig på att porta resten av GapiDraw i den ordning som följer:

- CGapiDisplay
- CGapiSurface
- CGapiInput
- CGapiBitmapFont

Porta GapiDraw till Palm OS

Strategin var att arbeta med en liten del av koden och försöka få den att fungera och sen fortsätta med att lägga till mer och mer kod. Så det första problemet var att få tillgång till skärmen för att kunna köra funktionen OpenDisplay i klassen CGapiDisplay.

För att rita till skärmen finns det två sätt i Palm OS. Det ena, som Palm rekommenderar, är att skapa ett fönster i det dynamiska minnet med WinCreateBitmapWindow och sen sätta det nya fönstret som aktivt. Detta tar alldeles för lång tid och utvecklaren vill ha tillgång till pekaren till först biten så att utvecklaren själv kan bestämma var och vad han vill rita. Detta är det andra sättet, men det rekommenderas inte av Palm därför att det är lätt att göra fel, t.ex. att skriva utanför minnet. Detta gör man via en funktion som heter WinScreenLock. Denna funktion ”fryser” skärmen och returnerar pekaren till ett minnesutrymme i VRAM:et som är exakt lika stort som det minne som är till skärmen. När man har det

minnesutrymmet kan man konstruera vad man vill rita till skärmen och efter det kopierar man tillbaka det minnesutrymmet till skärmen med en funktion som heter WinScreenUnlock. Eftersom att WinScreenLock skapar ett lika stort minnesutrymme i VRAM:et som skärmen har är detta en väldigt kostsam operation rent tidsmässigt därför så skapar jag ett minnesutrymme i det dynamiska minnet som är exakt lika stor som skärmen och som det extra minnesutrymmet i VRAM:et och gör alla ritoperationer eller blittar till det dynamiska minnet och när jag är klar kopierar jag bara över minnet till VRAM:et och sen kopierar jag över detta till skärmens minnesutrymme. Detta gjorde jag därför att skriva till det dynamiska minnet var snabbare än att skriva till VRAM:et och på så sätt blev alla operationer snabbare. Denna teknik kallas dubbelbuffring och används på många plattformar, inte bara på Palm OS. Att man ”fryser” skärmen och skapar ett temporärt minneutrymme gör man därför att om man skulle rita direkt till skärmen så skulle användaren av applikationen uppfatta skärmen som att den flimrar.

När man väl fått fram pekaren till första byten i minnesutrymmet för skärmen så behöver man veta hur stora x-Pitch och y-Pitch är. X-Pitch är hur många bytes i minnet man behöver stega för att komma till nästa pixel i sidled och y-Pitch är då hur många byte i minnet man behöver stega för att koma till nästa pixel i höjdled. För att ta reda på dessa för Pocket PC så använder GapiDraw sig av GAPI som ger pekaren till skärmen och x-Pitch och y-Pitch, men eftersom inte GAPI finns för Palm OS så var jag tvungen att använda någon av de inbyggda funktionerna i Palm OS. Den funktionen jag använde var WinScreenMode tillsammans med WinScreenGetAttribute. WinScreenMode känner av vilken enhet det är och vilken skärmtyp den har både upplösning och färgdjup, dessutom får man storleken i bredd och höjd. Därefter för att kunna hämta dessa värden så använde jag WinScreenGetAttribute, där man specificerar vilket värden man vill få ut t.ex. så vill man hämta bredden så använder man sig av WinScreenGetAttribute(winScreenWidth, <variabel att spara värdet i>). För att få ut x-Pitch så antog jag att de ligger efter varandra i minnet vilket betyder att för att komma till nästa byte i sidled borde man stega så många byte som färgdjupet är. För att komma åt y-Pitch så fanns det ett värde som kallades winScreenRowBytes, alltså antal byte för en rad på skärmen vilket är exakt vad jag vill ha, så det fick bli min y-Pitch. Jag hade även kunnat använda mig av (x-Pitch * bredden i pixlar), men eftersom att det fanns en inbyggd funktion som gav mig det jag villa ha direkt så använde jag mig av den. När det gäller färgdjup så stödjer GapiDraw bara färgdjup på två bytes, vilket betyder 16 bitar. Dessa bitar fördelar sig som antingen 565, 555, eller 444 vilka är de tre format som GapiDraw stödjer. När det gäller Palm OS så körs de nya enheterna på en ARM processor, men operativsystemet är bakåtkompatibelt med Motorolas 68k processor vilket gör att ARM processorn har stöd för 565 Little Endian och 68k processorn har stöd för 565 Big Endian. Detta medförde att för varje pixel med färg som en utvecklare vill rita ut på skärmen oavsett om det är i en bild eller ett streck eller bara en pixel så måste den färgen konverteras till Little Endian. Detta tog väldigt lång tid att komma på vad det var för fel och åtskilliga timmar att utveckla något smart som gjorde att utvecklarna aldrig skulle behöva fundera på detta. Den första strategin var att skriva om färgkonverteringsrutinerna så att GapiDraw skulle använda olika rutiner för Palm respektive Pocket PC. Detta gav väldigt mycket att tänka på och mycket binärräkning, men tillsist fick vi det att fungera. Men efter att ha läst en hel del dokumentation och frågat en

del på forumet så kom jag fram till att det fanns en rutin i Palm OS som kallades ByteSwap som kunde användas. Detta blev en mycket snyggare lösning därför att jag inte behövde ha olika färgkonverteringsmetoder för olika plattformar utan det räckte att köra en ByteSwap på pixeln innan den skickades till färgkonverteraren.

Porta XKobo till Palm OS

För att se om det gick att lyfta över ett spel från en plattform till en annan och se om applikationen skulle vara spelbar på båda plattformarna, d.v.s. utan att hacka fram och att flyta på i rörelserna för både fiender och spelare, dessutom klara av att få en mjuk scrollning av bakgrunden, skulle jag skriva en applikation som kunde köras på både Pocket PC och Palm OS. Eftersom att jag hade portat grafikbiblioteket GapiDraw till Palm OS så passade det bäst med att porta ett spel som var skrivet med hjälp av GapiDraw. Min handledare tipsade mig om ett spel som heter XKobo som redan var portat till Pocket PC och heter då PocketKobo. XKobo är skrivet av Akira Higuchi och han skrev spelet till Unix och X Windows. XKobo är ett single-player spel där man kan flyga i alla olika riktningar och förstöra olika labyrinter som skjuter mot dig, dessutom möter man olika fiendeskepp, stenar och andra typer av fiender som gör allt för att du inte ska lyckas ta dig till nästa bana. XKobo blev portat av Leif Kornstaedt till Pocket PC och han lade även till ljudeffekter. Anledningen till att jag valde att porta detta spel var dels att det är ett klassiskt spel som har spelats av många människor på både stationära datorer och Pocket PC och fått väldigt höga betyg och dels för att spelet redan var portat till Pocket PC så att jag inte behövde skriva ett spel först till Pocket PC och sen porta det till Palm OS. Det hade tagit alldeles för lång tid för att falla inom ramen för denna uppsats. Dessutom är PocketKobo skrivet i så kallad OpenSource, vilket betyder att källkoden är fri och vem som helst kan ladda ner den. Detta sparade jag väldigt mycket tid på och var nödvändigt för att jag skulle kunna hinna med att porta ett spel till Palm OS.

Eftersom att PocketKobo skrevs under 2002 så använde Leif, GapiDraw version 1.01 när han portade PocketKobo och den version jag har portat är version 2.05. Detta gjorde att det krävdes en hel del förändringar i koden för att få PocketKobo att fungera på Pocket PC. Det första jag gjorde var att anpassa alla funktionsanrop till version 2.05. Det hade tillkommit en del valmöjligheter och optimeringar till vissa funktionsanrop och därför behövde koden anpassas. Det andra jag fick göra var att inkludera några filer som användes lite olika i GapiDraw 1.0 och GapiDraw 2.05. Den stora skillnaden mellan Pocket PC och Palm OS är Palm OS begränsning när det gäller att förflytta sig i minnet. Palm OS kan inte använda sig av kodsegment större än 64 kbyte vilket gjorde att jag fick skriva om alla banor, därför att de var deklarerade som statiska variabler och skapades därför på stacken där det inte finns så mycket utrymme i mobila enheter, så även i Palm OS. Att kodsegmenten inte får vara större än 64 kbyte är för att Palm OS stödjer Motorolas 68k processor och den kunde bara adressera adresser i minnet upp till 64 kbyte. D.v.s. att processorn inte klarar av att förflytta sig över större minnes area än 64 kbyte. För att kunna få plats med dessa banor skapade jag dem dynamiskt och på så sätt skapade kompilatorn banorna i det dynamiska minnet när banan behövdes. Genom att skapa en pekare och sedan skapa minnesutrymme med *malloc*

så lade jag banorna i det dynamiska minnet istället för på stacken. Jag fick även skriva om key.cpp för att hantera knapptryckningarna korrekt i Palm OS.

Figur 11. PocketKobo på både Toshiba e740 som kör Pocket PC och Zire 71 som använder sig av Palm OS, portat med hjälp av GapiDraw 2.05.

Evaluering

I detta kapitel behandlar jag evalueringen av mitt arbete. Jag tar upp syftet med evalueringen och går igenom steg för steg tillvägagångssättet för detta arbete och vad jag har kommit fram till för resultat under arbetets gång.

Syfte

Syftet med evalueringen är att besvara mina forskningsfrågor:

- *Går det att utveckla spel med en gemensam kodbas för mobila handhållna enheter som använder olika operativsystem?*
- *Om ja, krävs det några specifika verktyg och plattformar för att genomföra detta?*
- *Om ja, hur bör en sådan applikation designas och implementeras?*

För att svara på dessa frågor har jag tagit reda på skillnader mellan olika operativsystem, för att se om det var någon övergripande skillnad i flera operativsystem. Dessutom har jag portat stora delar av grafikbiblioteket GapiDraw och portat ett spel från operativsystemet Pocket PC till Palm OS, för att se om det är möjligt att med hjälpa av GapiDraw kunna flytta över olika spel mellan olika plattformar. För att få en uppfattning om hur mycket det skiljer mellan olika funktioner på olika plattformar har jag gjort en benchmark över de mest använda funktionerna vid spelutveckling.

GapiDraw

När det gäller grafikbiblioteket GapiDraw så har jag portat de delar som jag tillsammans med min handledare kom fram till att jag skulle porta. Dessa delar är CGapiSurface, CGapiDisplay, CGapiBitmapFont samt CGapiInput. CGapiRGBASurface har inte portats då det jag hade behövt skriva en bildinläsningsrutin som stödjer variabel alphainformation och det hade tagit för lång tid för att falla inom ramen för detta examensarbete. CGapiCursor portades inte heller då det inte finns några pekdon för handdatorer och dessutom är CGapiCursor en underklass till CGapiRGBASurface. På grund av dålig noggrannhet i Palm OS portades inte heller CGapiTimer. CGapiTimer förutsätter att man kan få operativsystemet att ”sova” i en millisekund men Palm OS kan endast garantera något mellan 1-10 millisekunder. CGapiApplication blev inte heller portat på grund av tidsbrist.

Plattformsoberoende

Då inte CGapiApplication är portat går det inte att flytta över ett spel som är skrivet för Pocket PC till Palm OS, men samtliga funktionsanrop fungerar lika till Pocket PC plattformen som till Palm OS vilket gör att det bara är själva spelloopen som skiljer sig och möjligheten att minimera sin applikation. Att skriva CGapiApplication är inte alltför tidsödande och kommer att göras utanför ramen för detta examensarbete och spelloopen

finns och dessutom funktionaliteten att minimera en applikation så det är fullt möjligt att få den funktionalitet som CGapiApplication har på Pocket PC.

Vad jag har kommit fram till att man ska tänka på när man utvecklar applikationer till mobila enheter för att kunna få dem plattformsoberoende är att lägga så lite data som möjligt på kodstacken. Vad det gäller Palm OS begränsningar med 64 kbyte segment så kommer det att förbättras med det nya operativsystemet. Palm kör redan på ARM teknologi men strävar efter att kunna vara bakåtkompatibla med tidigare Motorola teknologi och därför har en begränsning på att bara kunna förflytta sig 64 kbyte i minnet. Vidare när man konstruerar program som man vill att de ska fungera på flera olika plattformar så ska man passa sig för globala variabler, då det inte finns stöd för dessa i flera operativsystem. När det gäller att utveckla applikationer till Palm OS bör man tänka på att det inte finns något filsystem. Detta ger att om man vill ha med bilder i sin applikation så bör man inkludera dess som resurser. Om man föredrar att inkludera bilderna via fil så kräver detta att användaren av applikationen har en enhet som har stöd för SD-kort och att användaren har ett sådant med plats för att lagra dessa bildfiler. Detta sätter krav på användaren vilket blir en begränsning om man vill sälja sin applikation eller sprida den vidare.

Benchmark

För att få lite siffror på prestandan av GapiDraw för Palm OS utförde jag en Benchmark, så att jag kunde jämföra GapiDraw för Pocket PC och GapiDraw för Palm OS. Benchmarken gjorde jag genom att ta reda på vilka funktioner som användes mest under spelutveckling. Detta gjorde jag genom att ta hem 25 olika spel och testa dem för att se vilka funktioner de använder, för att se hela listan se Appendix A. Jag kom fram till att de var främst fem funktioner som användes i de flesta spel. Dessa var Blt, BltFast, DrawText, FillRect och SetPixel. För att testa dessa gjorde jag en del olika varianter med de olika möjligheterna som erbjuds, för hela benchmarken se Appendix B.

5000 blits	**Stationary PC**	**PocketPC 2002**	**PalmOS 5**
	GD205	GD205	GD205
Starting with default display mode			

BltFast 64x64	35	328	1230
BltFast 64x64 with 80% opacity	298	2098	7590
BltFastColorkey 64x64	75	370	6290
BltFastColorkey 64x64 with 80% opacity	113	767	8290
FillRect 64x64	22	276	6010
FillRect 64x64 with 80% opacity	271	1882	5050

100000 blits			

DrawLine VLine 200px	165	13989	39905
DrawLine Diagonal 200px	180	14098	77822

Resultatanalys

Idag används GapiDraw till en mängd olika speltitlar för handdatorer. Många av dessa titlar har fått priser som

- Arcade Game of the Year 2003 runner up, Toki Tori
- RPG Game of the Year 2003, Everquest 3
- Sims Game of the Year 2003, Tower Mogul
- Strategy Game of the Year 2003 runner up, Warlords 2

Även stora speltillverkar så som SONY använder sig av GapiDraw när de utvecklar sina spel, exempel på detta är EverQuest som även har vunnit pris som bästa RPG spel för handdatorer. GapiDraw är en välanvänd plattform när det gäller utveckling av spel till mobila plattformar. För att testa om det går att lyfta över ett spel utvecklat med GapiDraw till Palm OS, så gjorde jag en benchmark där jag jämförde tiden det tog att utföra ett förutbestämt antal repetitioner av de olika funktionerna som används för spelutveckling och det jag kom fram till var att det går att utveckla spelapplikationer till Palm OS med grafikbiblioteket GapiDraw och att lyfta över redan utvecklade spel till Palm OS plattformen. För att verkligen vara säker på att detta fungerar portade jag ett spel själv från Pocket PC till Palm OS med hjälp av GapiDraw version 2.05 och ser om spelet *flyter* önskvärt. Det jag kom fram till efter att ha portat PocketKobo var att spelet är något långsammare på Palm OS än Pocket PC versionen men är helt klart spelbart utan att *hacka fram.*

Vad det gäller utvecklingsverktyg så är det Metrowerks CodeWarrior som gäller där utvecklaren själv kan ställa in de inställningar som behövs för att dels få sin egen kod att fungera men samtidigt att få GapiDraw att fungera. För att få sin spelapplikation att fungera bör man tänka på att Palm OS har begränsningen att inte kunna förflytta sig längre än 64Kbyte i minnet, vilket gör att när man utvecklar sin spelapplikation behöver man designa denna så att kodsegmenten aldrig överstiger 64Kbyte. Den kritiska delen för just detta problem är när man skapar sin värld och sina spelare samt de motståndare man vill ha med i sin spelapplikation. Lösningen till detta är att skapa dem dynamiskt så att de hamnar i det dynamiska minnet.

Slutsatser och Framtida arbete

Under detta kapitel sammanfattar jag de resultat som jag har kommit fram till under denna uppsats och tar även upp framtida arbete, hur man kan fortsätta med detta arbete

För att sammanfatta de resultat jag har kommit fram till svarar jag på de forskningsfrågor jag ställt i början av rapporten:

- *Går det att utveckla spel med en gemensam kodbas för mobila handhållna enheter som använder olika operativsystem?*
- *Om ja, krävs det några specifika verktyg och plattformar för att genomföra detta?*
- *Om ja, hur bör en sådan applikation designas och implementeras?*

Går det att utveckla spel med en gemensam kodbas för mobila handhållna enheter som använder olika operativsystem?

Det jag har kommit fram till är att det går alldeles utmärkt att utveckla spel som har en gemensam kodbas om man följer de regler som operativsystemet i sig kräver. Till exempel så med Palm är det ett krav att man håller sina kodsegment under 64 Kbyte och att använder sig av Palm OS 5 eller senare. Det skulle gå att anpassa kodbasen till att klara av tidigare versioner av Palm OS, men då hade vi fått skurit ner i funktionaliteten hos GapiDraw.

Om ja, krävs det några specifika verktyg och plattformar för att genomföra detta?

Det verktyg som jag har använt mig av är Metrowerks CodeWarrior och den måste man använda för att kunna utveckla spel till Palm OS med hjälp av GapiDraw. När det gäller grafikbibliotek så har jag koncentrerat mig på att utveckla GapiDraw och anpassat det biblioteket till att användas för utveckling av mobila spel till multipla plattformar. Enligt mitt tillvägagångssätt med denna uppsats så skulle det inte hindra att ha använt ett annat grafikbibliotek, under förutsättning att koden finns och att biblioteket fungerar på liknande sätt. Även när det gäller operativsystem så ser jag ingen som helst begränsning att lyfta över GapiDraw till en annan plattform och få även den plattformen att kunna använda spel gjorda med GapiDraw.

Om ja, hur bör en sådan applikation designas och implementeras?

När det gäller designen av spel till multipla mobila operativsystem så får man tänka på att mobila enheter har begränsat med minne, de har mindre antal färger och mindre display. Detta är saker som är väldigt viktiga när man utvecklar applikationer till mobila enheter dessutom ska man passa sig för att ha med operativsystemsberoende funktionsanrop, därför att det finns kanske ingen motsvarighet i andra operativsystem. Även globala variabler ska

man försöka undvika då det fyller upp stacken ganska fort på mobila enheter, därför att minnet är ytterst begränsat. Personligen tycker jag även att man ska undvika för mycket detaljer då den mobila enhetens skärm är väldigt begränsad och applikationen kan uppfattas som ”plottrig” om det blir för mycket detaljer på en för liten yta.

Framtida arbete

När det gäller fortsatt arbete på detta arbete så behöver man knyta ihop allting till någon form av ett dynamiskt länkat bibliotek. Det finns två varianter i Palm OS, den ena är Shared Library och den andra är Static Library. Shared Library är ett bibliotek som man kan använda olika funktioner ur och liknar mest PC:s motsvarighet DLL och Static Library är ett bibliotek som länkas in i applikationen statiskt. Vad som är bäst ingår inte i detta arbete för mig att avgöra. Det som också behöver göras är att skapa CGapiApplication så att utvecklaren får tillgång till spelloopen m.m. Det som har kommit fram efter arbetet är klart är att man skulle tjäna en hel del i prestanda om man istället för att använda sig av MemMove vid kopiering använde sig av WinCopyRectangle som är ett av Palm OS funktionsanrop. Det blev en markant prestanda höjning med WinCopyRectangle men då detta upptäcktes efter att utvecklingen var klar så faller det inte inom ramen för denna uppsats utan får bli ett senare projekt.

Referenser

1. Jonathan P. Allan(1999), *What went wrong?,* **Handheld and Ubiquitous Computing: First International Symposium, HUC'99, Karlsruhe, Germany, September 1999**
2. Anders Nordner och Henrik Svidén (2004), *"Nöjesindustrin blir mobil"*, **Computer Sweden nummer 32, fredagen den 19 mar -04. sid 10**
3. IDC Press Center (2004), http://www.idc.com/nordic/press/20040123.htm **040123**
4. Jonas Linderoth(2002), "Eleven, Leken och Spelet", **ITiS Delegationen för IT i Skolan, 2002**
5. Kurt Squire(2002), "Cultural Framing of Computer/Video Games", **Game Studies – the international journal of computer game research, volume 2, issue 1, Juli 2002**
6. Stina Nylander, Markus Bylund och Magnus Boman (2004), *Mobile Access to Real-Time Information – the case of autonomous stock brokering*, **Personal and Ubiquitous Computing pp 42-46, Februari 2004**
7. Bakhtiar Mikhak, Fred Martin, Mitchel Resnick, Robbie Berg and Brian Silverman (1999), *The Children´s Machines: Handheld and Wearable Computers Too*, **Handheld and Ubiquitous Computing: First International Symposium, HUC'99, Karlsruhe, Germany, September 1999**
8. Johan Sanneblad och Lars Erik Holmquist (2002), *Prototyping Mobile Games*, **IWEC2002**
9. DirectX, http://www.microsoft.com/directx, **040402**
10. Dan Bendas och Mauri Myllyaho (2002), *Wireless Games*, **Product Focused Software Process Improvement : 14th International Conference, PROFES 2002 Rovaniemi, Finland, December 9-11, 2002.**
11. Alexandra Weilenmann (2001), *Negotiating Use: Making Sense of Mobile Technology*, **Personal and Ubiquitous Computing, July 2001**
12. Jonas Bylund och Erik Ruisniemi (2004), *GapiDraw för Symbian,* **Department of Applied Information Technology, IT-Universitetet, Februari 2004**
13. GapiDraw, www.gapidraw.com **040327**
14. MobileCore, http://sourceforge.net/projects/mobilecore/ **040327**
15. Mophun, www.mophun.com **040327**
16. PocketFrog, http://pocketfrog.droneship.com/ **040327**
17. EG Components, http://www.egcomponents.se/include/meteoren/meteorendec02.pdf **040320**
18. Nocom årsredovisning 1998 (1999), www.nocom.com **040320**
19. 1-UP top 5 bästa spelen, http://www.1up.com/category2/1,4362,1227409,00.asp **040314**
20. Nintendos Hemsida, http://www.nintendo.com/home **040225**
21. WarCraft Release, http://www.mobygames.com/game/versions/gameId,371/ **040314**

22. WarCraft II Release, http://www.mobygames.com/game/versions/gameId,1339/ **040314**
23. Half-Life Release, http://www2.uiah.fi/~taharvia/halflife/history.htm **040314**
24. Counter-Strike Release, http://www.mobygames.com/game/versions/gameId,2726/ **040314**
25. StarCraft Release, http://www.mobygames.com/game/versions/gameId,378/ **040314**
26. GameSpot History, http://www.gamespot.com/gamespot/features/video/hov/p2_02.html **040225**
27. Teixeira de Sousa, B.M. (2002), *Game Programming All in One*, **Premier Press, Inc, USA**

Appendix A

Här listas de spel upp som jag testade under min testperiod för att få fram vilka funktioner som användes oftast i spel utveckling. Anledningen till att det blev just dessa spel är att de är de spel som har fått högst användarbetyg på www.handago.com. Vilka funktioner som används fast ställdes genom att testa spelen och jämföra grafikoperationerna med GapiDraw API.

Games for PocketPC

Simple Demo: BltFast, FillRect, FillRect med opacity, DrawText, DrawLine, Blt
Volleyball: Blt, BltFast, BltFast med opacity, FillRect, DrawText
Animals DX: BltFast, BltFast med opacity, FillRect, DrawText
Basketball: BltFast, DrawText, FillRect, FillRect med opacity, DrawLine, DrawRect
Candy Catchup: BltFast, BltFast med opacity, DrawText, FillRect
Chopper Rescue: Blt, BltFast, BltFast med opacity, DrawText, FillRect
EverQuest: Blt, BltFast, FillRect, DrawLine, DrawRect, DrawText
Escape Velocity: BltFast, BltFast med opacity, DrawText, FillRect med opacity, DrawRect
Fortune Taipei: Blt, BltFast, DrawLine, DrawText, FillRect
Galactic Assault: BltFast, BltFast med opacity, DrawText, DrawRect, FillRect
Kings Crown: BltFast, BltFast med opacity, FillRect, DrawRect, DrawLine, DrawText
Lines3D: BltFast, DrawLine, DrawText, FillRect
Mahjong Dragon: BltFast, DrawText, DrawRect, FillRect
Mini Golf Challenge: BltFast, FillRect, FillRect med opacity, DrawText
Moonlander: BltFast, FillRect, DrawText, SetPixel
MouseMaze: Blt, BltFast, DrawText, FillRect, DrawRect, DrawLine
PocketRocket: BltFast, BltFast med opacity, DrawText, FillRect, DrawLine
Railroad Jam: BltFast, DrawLine, FillRect, DrawText, DrawRect
Rainy Colors: Blt, BltFast, DrawText, FillRect, BltFast med opacity, DrawRect, DrawLine
Traffic Jam: BltFast, FillRect, DrawText, DrawLine
TommyK: BltFast, BltFast med opacity, Blt, DrawText, DrawRect
Tower Mogul: BltFast, FillRect, FillRect med opacity, DrawText, DrawLine
Triple Tower: BltFast, DrawRect, FillRect, DrawText
TubeMix: BltFast, BltFast med opacity, FillRect, FillRect med opacity, SetPixel, DrawText
WarLords II: BltFast, FillRect, DrawText
WordChallenge: BltFast, BltFast med opacity, FillRect, DrawText

Appendix B

Här visar jag resultatet av den benchmark som jag gjorde over de funktioner jag kom fram till användes oftast vid spelutveckling. Se Appendix A.

5000 blits	**Stationary PC**	**PocketPC 2002**	**PalmOS 5**
	GD205	GD205	GD205
Starting with default display mode			

BltFast 16x16	10	49	480
BltFast 16x16 with 80% opacity	17	154	2370
BltFast 32x32	15	115	710
BltFast 32x32 with 80% opacity	80	547	3340
BltFast 64x64	35	328	1230
BltFast 64x64 with 80% opacity	298	2098	7590
BltFast 128x128	165	3694	8020
BltFast 128x128 with 80% opacity	1164	19653	26310
BltFastColorkey 16x16	10	56	2380
BltFastColokey 16x16 with 80% opacity	15	103	2490
BltFastColorkey 32x32	27	142	2880
BltFastColorkey 32x32 with 80% opacity	53	332	3130
BltFastColorkey 64x64	75	370	6290
BltFastColorkey 64x64 with 80% opacity	113	767	8290
BltFastColorkey 128x128	265	4698	18790
BltFastColorkey 128x128 with 80% opacity	453	8780	23430
FillRect 16x16	5	25	890
FillRect 16x16 with 80% opacity	20	149	2360
FillRect 32x32	5	88	2050
FillRect 32x32 with 80% opacity	70	498	2910
FillRect 64x64	22	276	6010
FillRect 64x64 with 80% opacity	271	1882	5050
FillRect 128x128	70	1127	20290
FillRect 128x128 with 80% opacity	1052	13332	14260

100000 blits			

DrawLine VLine 10px	22	216	11997
DrawLine VLine 100px	90	873	24957
DrawLine VLine 200px	165	13989	39905
DrawLine HLine 10px	13	155	12210
DrawLine HLine 100px	30	263	25182
DrawLine HLine 200px	40	392	39570
DrawLine Diagonal 10px	30	251	15712
DrawLine Diagonal 100px	100	908	44807

DrawLine Diagonal 200px	180	14098	77822

Rotate the display 90 degrees clockwise			

BltFast 16x16	5	51	490
BltFast 16x16 with 80% opacity	20	158	2390
BltFast 32x32	10	115	710
BltFast 32x32 with 80% opacity	75	543	3370
BltFast 64x64	40	330	1310
BltFast 64x64 with 80% opacity	290	2140	7650
BltFast 128x128	160	3741	8090
BltFast 128x128 with 80% opacity	1144	20439	27110
BltFastColorkey 16x16	7	49	2430
BltFastColokey 16x16 with 80% opacity	10	94	2520
BltFastColorkey 32x32	20	127	2950
BltFastColorkey 32x32 with 80% opacity	40	312	3310
BltFastColorkey 64x64	83	401	6350
BltFastColorkey 64x64 with 80% opacity	130	811	8410
BltFastColorkey 128x128	265	4782	19210
BltFastColorkey 128x128 with 80% opacity	511	9140	24010
FillRect 16x16	5	26	870
FillRect 16x16 with 80% opacity	10	149	2320
FillRect 32x32	5	90	2110
FillRect 32x32 with 80% opacity	65	504	3020
FillRect 64x64	20	278	6040
FillRect 64x64 with 80% opacity	260	1895	5160
FillRect 128x128	68	1250	20880
FillRect 128x128 with 80% opacity	1024	13598	14690

100000 blits			

DrawLine VLine 10px	10	154	12007
DrawLine VLine 100px	30	263	25237
DrawLine VLine 200px	40	386	40505
DrawLine HLine 10px	20	217	12970
DrawLine HLine 100px	83	874	25367
DrawLine HLine 200px	160	14127	39347
DrawLine Diagonal 10px	30	255	15440
DrawLine Diagonal 100px	92	914	45330
DrawLine Diagonal 200px	170	14189	78672

Rotate the display 180 degrees			

BltFast 16x16	5	53	500
BltFast 16x16 with 80% opacity	20	153	2390
BltFast 32x32	10	116	690

BltFast 32x32 with 80% opacity	77	549	3310
BltFast 64x64	35	331	1250
BltFast 64x64 with 80% opacity	293	2126	7710
BltFast 128x128	160	3502	8110
BltFast 128x128 with 80% opacity	1146	20083	26030
BltFastColorkey 16x16	10	52	2420
BltFastColokey 16x16 with 80% opacity	10	99	2530
BltFastColorkey 32x32	25	142	2850
BltFastColorkey 32x32 with 80% opacity	50	330	3210
BltFastColorkey 64x64	70	387	6180
BltFastColorkey 64x64 with 80% opacity	112	773	8230
BltFastColorkey 128x128	273	4659	18230
BltFastColorkey 128x128 with 80% opacity	445	8892	22890
FillRect 16x16	5	26	900
FillRect 16x16 with 80% opacity	12	154	2370
FillRect 32x32	5	90	2050
FillRect 32x32 with 80% opacity	67	496	2920
FillRect 64x64	20	287	6060
FillRect 64x64 with 80% opacity	260	1899	5130
FillRect 128x128	70	1046	21110
FillRect 128x128 with 80% opacity	1024	13736	14930

100000 blits			

DrawLine VLine 10px	20	222	12112
DrawLine VLine 100px	90	880	25127
DrawLine VLine 200px	160	14536	40010
DrawLine HLine 10px	10	154	12340
DrawLine HLine 100px	30	269	25542
DrawLine HLine 200px	40	391	39760
DrawLine Diagonal 10px	30	259	15830
DrawLine Diagonal 100px	100	916	45305
DrawLine Diagonal 200px	170	14698	78210

Rotate the display 90 degrees counter clockwise			

BltFast 16x16	5	50	480
BltFast 16x16 with 80% opacity	25	160	2380
BltFast 32x32	15	117	700
BltFast 32x32 with 80% opacity	82	547	3320
BltFast 64x64	37	331	1250
BltFast 64x64 with 80% opacity	327	2136	7650
BltFast 128x128	188	3513	7970
BltFast 128x128 with 80% opacity	1289	20141	26870
BltFastColorkey 16x16	7	49	2390
BltFastColokey 16x16 with 80% opacity	10	96	2460

BltFastColorkey 32x32	30	129	2900
BltFastColorkey 32x32 with 80% opacity	57	308	3210
BltFastColorkey 64x64	90	401	6410
BltFastColorkey 64x64 with 80% opacity	155	811	8530
BltFastColorkey 128x128	328	4786	19430
BltFastColorkey 128x128 with 80% opacity	586	9177	23860
FillRect 16x16	5	25	910
FillRect 16x16 with 80% opacity	15	152	2340
FillRect 32x32	5	91	2130
FillRect 32x32 with 80% opacity	78	498	2970
FillRect 64x64	25	287	6130
FillRect 64x64 with 80% opacity	288	1921	5240
FillRect 128x128	75	1273	22010
FillRect 128x128 with 80% opacity	1116	13755	14680

100000 blits			

DrawLine VLine 10px	15	154	11947
DrawLine VLine 100px	30	262	23867
DrawLine VLine 200px	47	390	39330
DrawLine HLine 10px	20	214	12305
DrawLine HLine 100px	95	894	25182
DrawLine HLine 200px	178	14688	39430
DrawLine Diagonal 10px	32	252	16340
DrawLine Diagonal 100px	110	924	45307
DrawLine Diagonal 200px	180	14788	78120