Utveckling av kortspelet Pidro i Flash: klientens funktionalitet

Ron Holmström

Examensarbete
Informationsteknik
2011

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	2779
Författare:	Ron Holmström
Arbetets namn:	Utveckling av kortspelet Pidro i Flash: klientens funktionalitet
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	Rundradion Ab - Svenska YLE

Sammandrag:
Målet med examensarbetets praktiska del var att utveckla en prototyp av kortspelet Pidro för uppdragsgivaren Rundradion Ab - Svenska YLE. De väsentligaste kraven var att spelet bör fungera i bläddraren med Adobe Flash och använda Rundradions inloggningsnamn som spelarnamn. Spelets utvecklingsteam bestod av tre personer varav arbetsuppgifterna var fördelare enligt spelmotor, datorspelare med artificiell intelligens och en klient för människospelare. Min del i projektet var att utveckla klienten.

Detta examensarbete behandlar utvecklingsarbetet av Pidro- klienten samt dess koppling till de övriga delarna av programutvecklingsprojektet. I den introducerande delen förklaras kort vad projektet går ut på och dess målsättningar. En kort introduktion till Flash samt olika verktyg som använts i projektet ges innan tyngdpunkten fästs vid projektets förverkligande.

Examensarbetet fäster också uppmärksamhet vid hur man skall planera utveckling av större programmeringsprojekt; Hur skall man gå till väga för att undvika att allt blir en enda röra?

Nyckelord:	Flash, ActionScript, klient-applikation, Flash Media Server, spel, Pidro, containers, delade object
Sidantal:	53
Språk:	Svenska
Datum för godkännande:	12.5.2011

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	2779
Author:	Ron Holmström
Title:	Development of card game Pidro: client side features and functions
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	Rundradion Ab - Svenska YLE

Abstract:
The objective of the project was to develop a prototype of the card game Pidro to our client Rundradion Ab - Svenska YLE. The game was supposed to work in the browser with Adobe Flash and use the client's own login system. The game was developed by three persons and the duties were divided according to game engine, computer player with artificial intelligence and a client for human players. My part of the project was to develop the client and that is what this thesis is about.

This thesis deals with the development of the Pidro client and its relation to the rest of the software development project. In the introducing part I briefly explain what the project is about and its goals. A short introduction to Flash and various tools used in project is given before the emphasis given to its implementation.

The thesis also pays attention to how planning of a large development programming project should be done; How to avoid everything becoming a big mess?

Keywords:	Flash, ActionScript, client application, Flash Media Server, game, Pidro, containers, shared objects
Number of pages:	53
Language:	Swedish
Date of acceptance:	12.5.2011

INNEHÅLL

Figurer

Förkortningar

AS	ActionScript
FMS	Flash Media Server
RCP	Remote Procedure Call
AI	Artificial Intelligence
SVN	Subversion
IP	Internet Protocol
RTMP	Real-Time Messaging Protocol
PHP	Hypertext Preprocessor

FÖRORD

Planeringen av detta examensarbete började redan på våren 2009 och programmeringsdelen av projektet gjordes till största del sommaren samma år. Efter det har det dock varit mycket testande och fixande, men själva skrivandet påbörjade jag först hösten 2010.

Projektet var mycket intressant och det gav mig en bra inblick i hur det är att programmera större projekt i samarbete med andra.

Jag vill tacka mina handledare Jonny Karlsson och Hanne Karlsson för all hjälp de gett mig i skrivandet av detta examensarbete. Jag vill också tacka det andra i utvecklingsteamet; Krister Bäckman och Peter Saloviin, för allt fint sammarbete och för ett lyckat projekt.

Till sist vill jag ännu passa på att tacka Rundradion Ab - Svenska YLE för möjligheten att delta i Pidro-projektet.

1 INLEDNING

1.1 Målsättning och syfte

Detta examensarbete behandlar utvecklingen av en Flash-klientapplikation som var en del av ett programutvecklingsprojekt beställt av Rundradion Ab - Svenska YLE. Syftet var att utveckla en prototyp av kortspelet Pidro och få det att fungera på Internet med flera samtidiga spelare.

Den färdiga prototypen är ett fungerande spel som kan publiceras på Rundradions webbplats och använder deras inloggningssystem. I prototypens programmeringskod har det tagits i beaktande möjligheterna för vidareutveckling och koden har därför kommenterats noggrant och gjorts så lättläst som möjligt.

Examensarbetet består av en introduktion till Flash och dess kompatibilitet för projektet, vilka verktyg som användes, hur projektet var förverkligat och slutligen vad resultatet blev.

Målsättningen för examensarbetet var att utveckla en prototyp för kortspelet Pidro i Flash och följa kravspecifikationen (Bilaga 1) som utarbetats tillsammans med beställaren.

1.2 Arbetsgruppen

I projektet jobbade vi i en arbetsgrupp på tre personer. Projektet delades därmed upp i tre olika delar; servern, klienten och datorspelaren. Min uppgift var att utveckla klienten. Utvecklingen av serverdelen beskrivs i examensarbetet ”Spelmotor för Rundradions Pidro-projekt” som är skriven av Krister Bäckman och datorspelaren i examensarbetet ”Utveckling av en testplattform för en datorstyrd spelare i kortspelet Pidro” skrivet av Peter Saloviin.

1.3 Begränsningar

Detta examensarbete behandlar inte Pidro-spelets regler. Vill man bekanta sig med spelreglerna kan dessa läsas i Bilagor

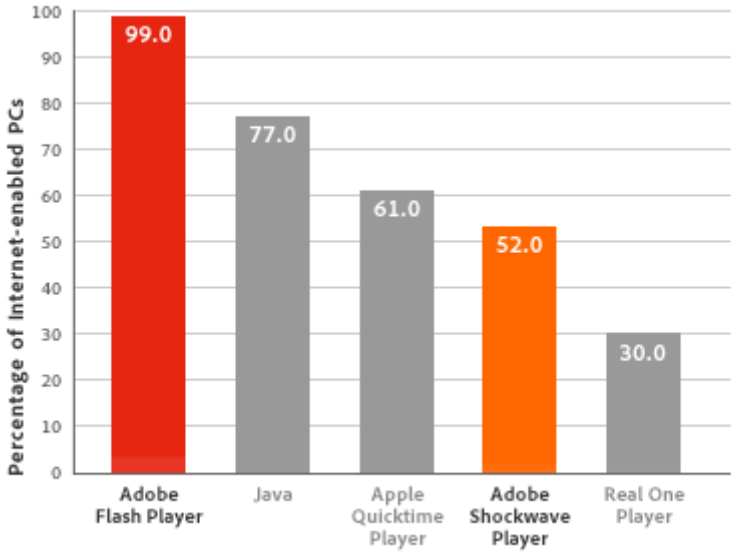
Bilaga 1 eller på följande portal pidro.net (Pidro Online, 2011).

Flash Media Server (FMS), som används som server i Pidro spelet, behandlas inte heller. Examensarbetet fokuserar sig enbart på utvecklingen av klienten

Examensarbetet tar upp programmeringslösningar också i form av kod och därmed rekommenderas att läsaren har kunskap i ActionScript 3 eller programmering i allmänhet.

2 FLASH

Adobe Flash som från början var Macromedia Flash är en multimediaplattform som används mest för att lägga till interaktivitet i webbsidor, oftast i form av reklam, animationer, spel, bild, ljud och andra så kallade ”Rich Internet Applications” (Adobe, 2010a). Den ursprungliga Flash-spelaren, som i huvudsak används i PC-miljö, har senare också spridit sig till mobila plattformer i form av en lättare version som heter Flash Lite. Fastän Flash måste installeras separat och inte kommer med operativsystemet eller alla Internet-bläddrare fanns Flash i mars 2010 redan installerat på 99 % av datorerna som har Internet kontakt, enligt statistiken på Adobes hemsida (Adobe, 2010b). Statistiken var ihopräknad från följande länder: U.S., Canada, U.K., Frankrike, Tyskland, Japan, Australien och Nya Zeland.



Figur 1: Statistik för användningen av Flash från mars 2010 (Adobe, 2010b).

2.1 Utvecklingshistoria

Programmeringsspråket som används för att utveckla Flash-applikationer kallas ActionScript (AS). AS var i början ett skriptspråk för Macromedias Flash ”authoring tool” och utvecklas nuförtiden av Adobe Systems som Adobe Flash. AS var i början mycket simpelt och hade endast några kommandon man kunde använda som kallades

för "actions". Dessa var ”Play”, ”Stop”, ”getURL” och ”goToAndPlay”. Dessa hänger ännu med i den nuvarande AS-versionen. År 1999 tog AS i bruk variabler, uttryck (expressions), operatorer, if-satser och loopar. Vid Flash version 4 fick AS officiellt namnet "ActionScript". 5 september 2000 lanserades Flash 5:an och då blev också AS version 1.0. Det var den första versionen av AS som man kunde påverka med JavaScript. Vid denna version kunde man skapa lokala variabler, egna funktioner som man kunde skicka parametrar till och ta emot ”return”-värden. I AS v1.0 kunde man också börja skriva AS i en vanlig textredigerare och kompilera koden istället för att bygga upp den med hjälp av Macromedias program.

Efter en tid kom Flash Player 6 och AS utvecklades vidare. Till användning kom ”switch” och ”strict equal”-operatorn (= = =), men en av de bästa egenskaperna i denna version var att AS fick redan i detta skede en prototyp av objektorienterad programmering och nyckelorden "class" och "object" kunde användas.

I september 2003 introducerades AS v2.0 i samband med Flash MX 2004. Den motsvarande Flash-spelaren hette Flash Player 7. I detta skede var AS redan färdig att användas för större och mer krävande applikationer. ”Class” och ”extends” var vanliga nyckelord i AS v2.0 och fastän AS v2.0 redan stödde en mera strukturerad objekt-orienterad programmeringsstil så blev koden ännu kompilerad med AS v1.0 ”bytecode”, som gjorde Flash också möjlig att köra i Flash Player 6.

AS v3.0 lanserades i juni 2006 och denna nya versionen av AS kompilerades nu med Adobe Flex v2.0 och hade sin motsvarande Flash Player 9.0.

Adobe Flex är en SDK (Software Development Kit) som används för att kompilera Flash-applikationer (Adobe, 2010c). AS v3.0 var så pass annorlunda jämfört med de tidigare versionerna att det gjordes en helt egen virtuell maskin i Flash Player 9 för att köra AS v3.0 koden. Flash Player 9 innehåller två olika virtuella maskiner; en som tolkar AS v3.0 och en annan för v1.0 och v2.0. Detta betyder att AS v3.0 kompilerar till en helt ny ”bytecode” som inte passar ihop med AS v1.0 och v2.0. I AS v3.0 togs det också i bruk stöd för ”package”, ”namespaces” och ”regular expressions”. I AS v3.0 introducerades också 3D-stöd.

Flash Player 10 som i början kallades för Astro, har ett mycket bättre stöd för 3D-grafik än föregående version. Flash Player 10 har bl.a. funktioner för rotering av 3D-grafikobjekt och en API (Application Programming Interface) för att rita i 3D.

Flash Lite, som varit en del av utvecklingen sedan Flash Player 4, är designat för små elektroniska enheter som t.ex. mobiltelefoner (Wikipedia, 2010a).

2.2 Anpassning till projektet

Adobe har fokuserat sig på att Flash skall fungera bra som RIA (Rich Internet Applications). Förutom detta fungerar Flash också som en klientapplikation. Eftersom ett av kraven för projektet var att produkten skulle kunna publiceras på en webbsida fanns det inte många plattformar att välja mellan. Som tidigare nämnts, ser man att Flash-plattformen är installerad på nästan alla datorer runt om i världen som har Internet kontakt. Förutom detta så innehåller Rundradions webbplats mycket Flash-applikationer från förut, så detta sågs inte som ett problem.

Flash lämpar sig även mycket bra för att göra spel på grund av dess effektiva sätt att hantera grafik och animering. Flash har också en mycket välutvecklad händelsehantering (event handling) som är väldigt viktigt speciellt i spel och även för att skapa ett bekvämt användargränssnitt. Händelser används för att kunna trigga en funktion när någonting händer t.ex. ett knapptryck. I Flash är detta viktigt i många fall, annars skulle programmeraren vara tvungen att göra funktioner som hela tiden körs och kollar ifall någonting har hänt. Detta skulle leda till att applikationen skulle kräva mycket datorkraft och koden skulle bli svårläst. En händelse kan egentligen placeras på vad som helst i Flash, både visuellt synliga objekt och inne i kodens variabler.

Det finns en mängd olika typer av händelsehantering i AS3. De viktigaste och mest använda i Flash-spel är sådana som lyssnar på mus-, tangentbords- och ”Enterframe” händelser. Enterframe är en händelse som triggar en funktion för varje visad bild i en filmsekvens i Flash. Vid behov kan man även skapa skräddarsydda händelser.

2.3 Vad är problemen med att göra projektet i Flash

En av de största problemen med att göra ett nätverksspel i Flash är att man måste hitta något att använda som server för att få kommunikationen att fungera bra och säkert mellan klienterna. Speciellt i situationer som i detta spel, Pidro, där man har fyra samtidiga spelare. Då måste interaktiviteten mellan servern och klienterna vara mycket snabb för att inte förstöra spelupplevelsen med långa väntetider. Andra problem som har att göra med Flash-applikationer i allmänhet är laddningstiderna. Ingen tycker om att vänta och detta kan orsaka att spelare lämnar spelet förrän det har laddats färdigt.

Ett problem som kan uppstå med projektet är funktionalitet mellan olika versioner av Flash-spelare. Alla använder inte på samma version och vissa kan ha riktigt föråldrade versioner, som spelet också skall fungera med.

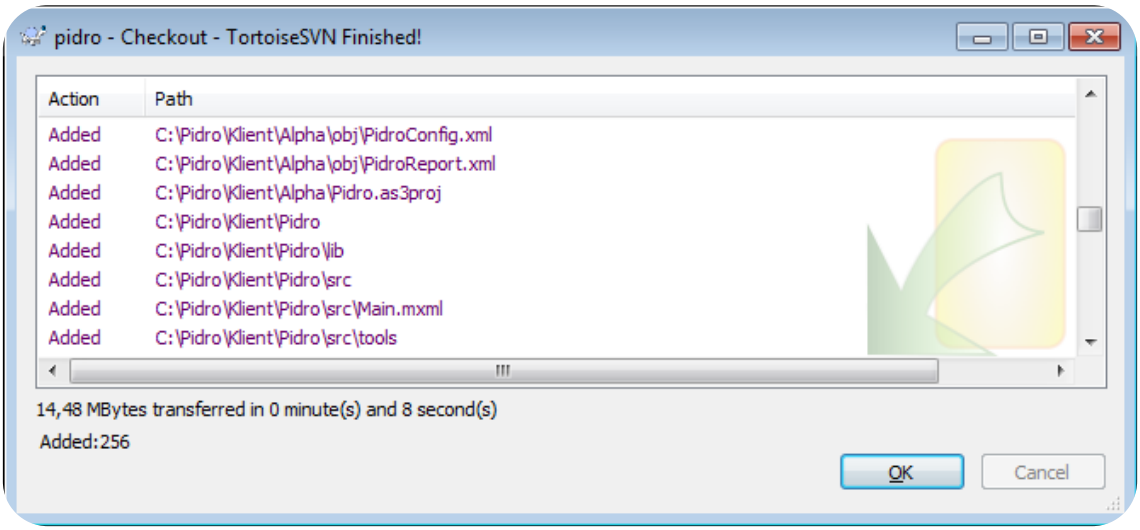
För att projektet skall lyckas och bli bra måste dessa problem tas på allvar och lösas redan i planeringsskedet.

3 VERKTYG

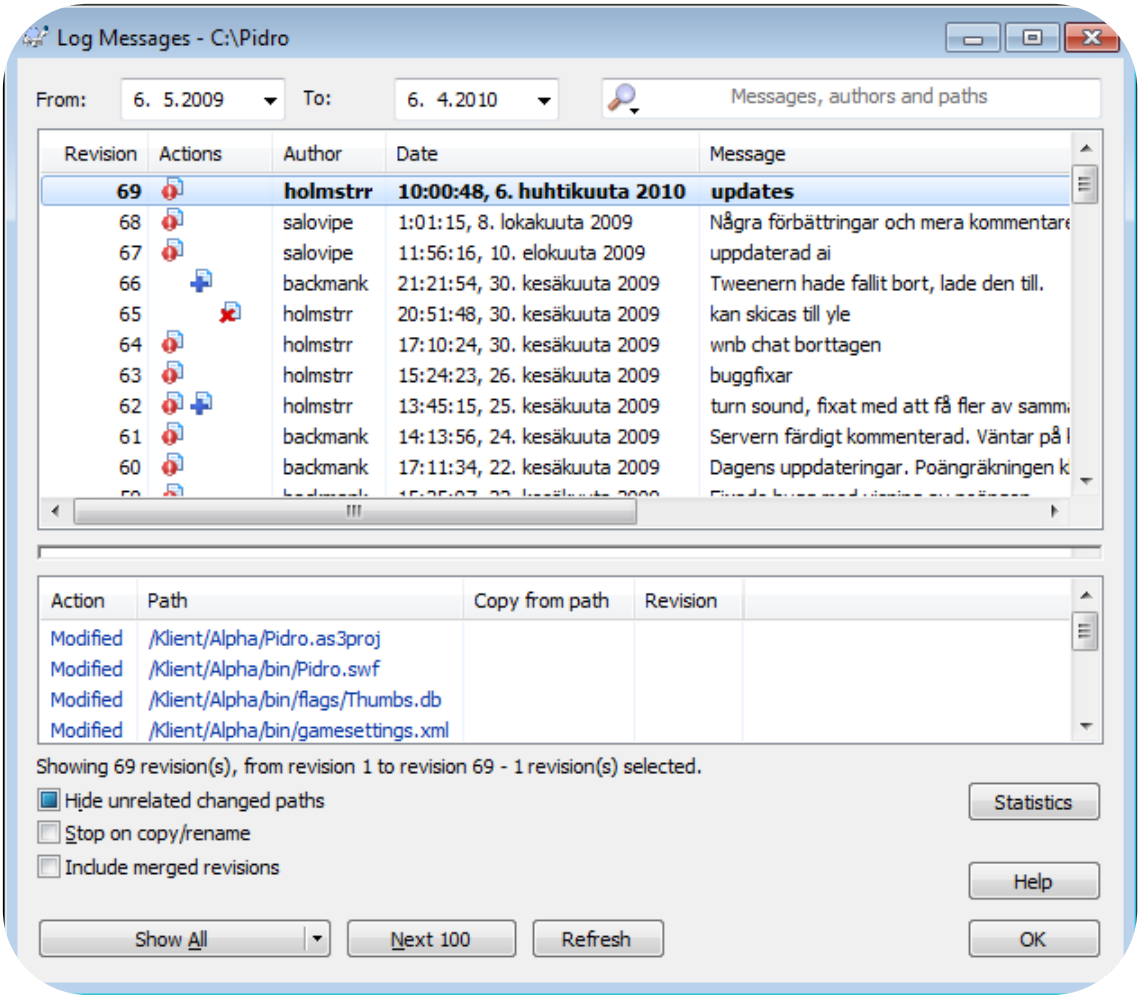
I detta kapitel följer en introduktion till de verktyg som användes i programutvecklingsprojektet. Det mest använda verktyget var Flashdevelop, som användes för att skriva programkoden. Andra program som användes på sidan om var Flash Media Server, som användes som server för kontakt mellan klienter. Även SVN, som är ett versionshanteringsverktyg, användes, och Mantis, användes som en samlingsplats för olösta problem i projektet. För projektet användes också en del icke-standard AS 3 bibliotek och några importerade bibliotek från Adobe Flash Professional.

3.1 SVN

SVN (Subversion) är ett versionshanteringsverktyg. Detta program används för att kunna hålla ordning och reda på olika versioner och milstolpar i ett programmeringsprojekt. Programmet är också mycket användbart när flera personer jobbar på ett projekt samtidigt. Alla versioner som laddas upp på SVN-servern sparas som en egen revision. Detta betyder att det är mycket lätt att komma åt äldre revisioner ifall någonting skulle ha gått på tok i den nuvarande eller nyaste revisionen av programmet (se Figur 3: Val av version av programmet man vill hämta). Man kan också bara ladda ner vissa delar av en revision. I revisionerna ser man också vem och när en person har laddat upp något och en revision kan också innehålla en manuellt skriven kommentar (Wikipedia, 2010b).



Figur 2: Checkout med TortoiseSVN



Figur 3: Val av version av programmet man vill hämta

3.2 Mantis

Mantis är ett gratis webbaserat felhanteringssystem skrivet i PHP (Hypertext Preprocessor). Mantis användes aktivt under hela projektets gång och var till väldigt stor nytta. Genast någon av programutvecklarna märkte en bugg eller hade en fråga laddade vi upp dem på Mantis. Det var inte bara projektarbetarna som hade tillgång till Mantis utan även Rundradion. Detta var också en mycket bra kontaktlänk till Rundradion och de tyckte om att följa med projektets framskridning och lösningar. Mantis hanterar buggar enligt status: nytt, återkoppling, godkänt, bekräftat, tilldelat, löst och stängt. Till en rapport i Mantis kan användarna lämna kommentarer, filer, ändra status eller tilldela problemet till en användare (Mantis, 2010).

Figur 4: Mantis framsida med tydlig information om det viktigaste

3.3 Flashdevelop

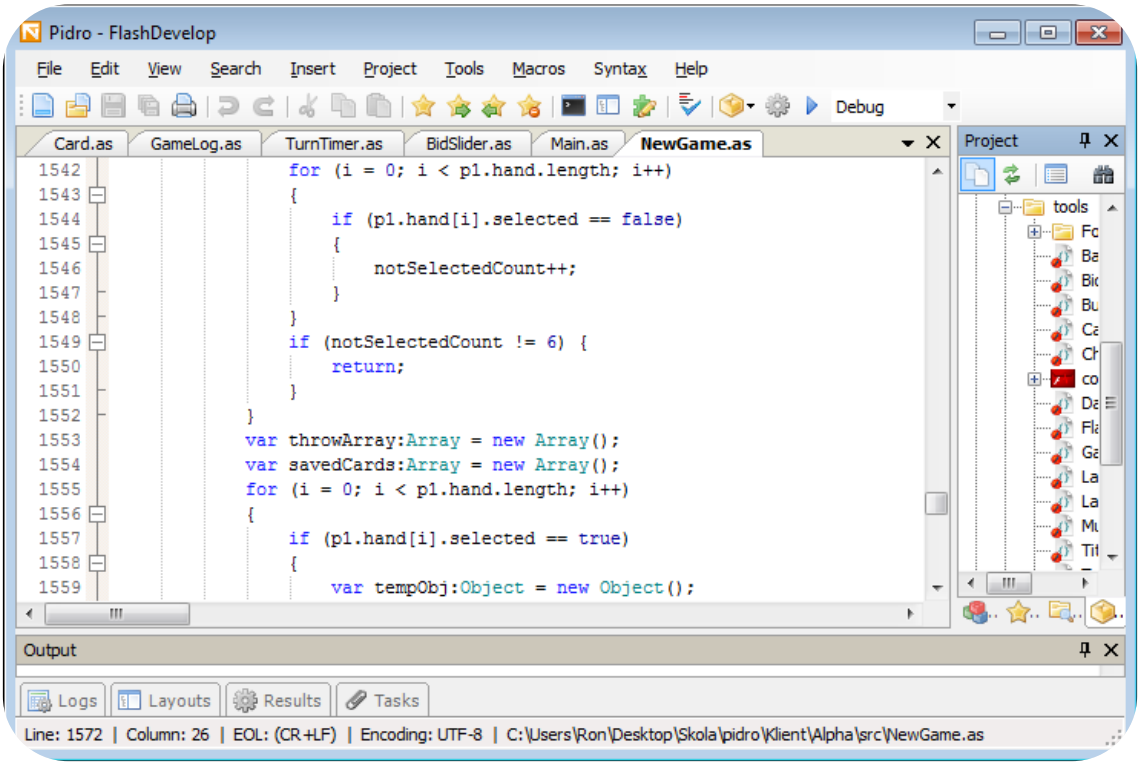
Flashdevelop användes som programmeringsverktyg i projektet. Programmet är enbart ett textbaserat programmeringsverktyg som saknar ”drag-and-drop”-funktioner och är också helt och hållet utan en tidslinje, som Adobe Flash användare är vana vid. Flashdevelop innehåller dock allting som behövs för att göra ett fullständigt projekt och har också en hel del extra ”plugins” som är möjliga att installera. Till skillnad från

Adobe Flash, är det lätt att bygga ett helt objektorienterat projekt i Flashdevelop tack vare dess projektlayout och struktur (se Figur 5: Flashdevelop, programmeringsverktyg).

Flashdevelops webbsida leder direkt till en wiki om Flashdevelop som gör det lätt att snabbt ladda ner programmet och hitta information om det (Flashdevelop, 2010a). För frågor och annat angående Flashdevelop, som inte hittas i wikin, har Flashdevelop ännu ytterligare ett forum. Under projektets gång använde jag mig av forumet för att få svar på en del frågor som uppstod under projektets gång (Flashdevelop, 2010b).

För att Flashdevelop skall kunna kompilera AS-kod, kräver den att Adobe Flex 3 SDK kompilatorn finns installerad på datorn och är länkad i Flashdevelops inställningar.

Flashdevelop kan även hantera flera andra programmeringsspråk förutom AS. Den har "syntax highlightning" och automatisk kodgenerering även för de andra språken.



Figur 5: Flashdevelop, programmeringsverktyg

3.4 TweenLite

TweenLite är en AS-klass som används för att animera på ett lätt och effektivt sätt. Det finns massor av olika animeringsklasser eller så kallade ”tweeners” och även Adobe har sin egna. TweenLite är gjort av ett företag som heter GreenSock. Detta företag har också en del andra klasser och plugins för AS 2 och AS 3 som kan hittas och laddas ner gratis från greensock.com (Greensock, 2010a). ”Tweenern” kräver ingen licensering ifall själva produkten den används i inte tar betalt av sina användare.

En ”tweeners” uppgift är närmast att hjälpa AS-programmeraren att skapa animationer utan att själv behöva skriva loop efter loop för att t.ex. flytta, snurra, förstora, förminska eller ändra genomskinligheten under ett visst intervall för ett visst objekt. Skulle man göra sina animeringar med egna loopar skulle det troligen leda till mycket svårläst och oorganiserad kod, om man inte förstås programmerar sin egen” tweener”-klass som skulle kräva stort arbete. Därför använder så gott som alla AS programmerare någon typ av ”tweener”-klass i sina projekt. Med en bra ”tweener” behövs det oftast bara en rad kod för att animera någonting. I koden skriver man vilket objekt som skall animeras, hur länge och vilka parametrar som skall ändra under tidsintervallen.

GreenSock erbjuder två olika versioner av ”tweeners”, TweenLite och TweenMax. TweenMax, som namnet också säger är en mera utvecklad version som har mer funktionalitet. Den viktigaste orsaken varför TweenLite användes i projektet istället för TweenMax är att den är mycket effektivt optimerad för prestanda.

”Tweenern” kan förutom grafisk animering också ”animera” olika siffervärden på vilket objekt som helst. Detta betyder att det inte behöver vara t.ex. en ”MovieClip” som animeras, utan det kan också vara ett ljuds volym som ”tweenas” eller en suddighetseffekt som ökar eller minskar suddigheten under en viss tid. TweenLite och GreenSock är ett aktivt projekt och företag och uppdateringar görs regelbundet, vilket är viktigt med tanke på vidareutveckling. Till projektet behövdes inga funktioner från TweenMax och TweenMax-klassen är en förlängning (extends) av TweenLite klassen, så detta var ett lätt beslut för prestandan. Skulle det i vidareutveckling behövas funktioner från TweenMax skulle allting fungera lika bra med TweenMax som med TweenLite (Greensock, 2010b).

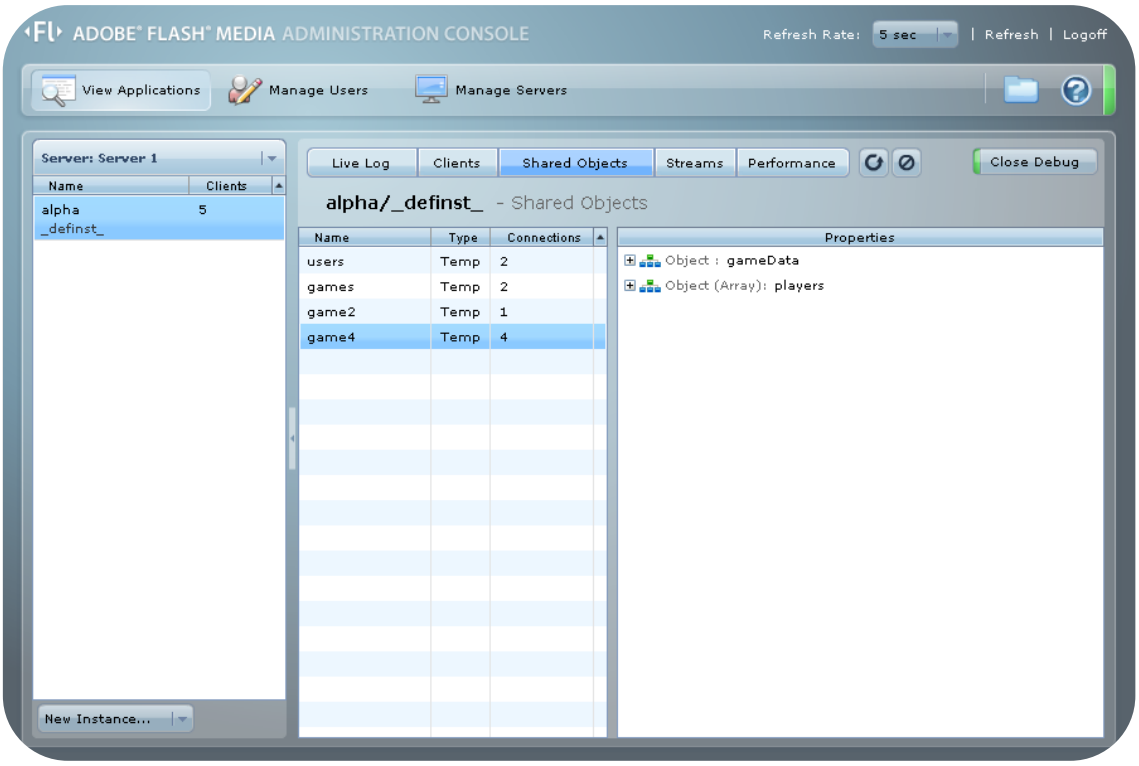
3.5 Flash media server

Adobes Flash Media Server (FMS) som från början var en Macromedia produkt kan installeras både på ett Windows- och Linux-operativsystem. Versionen av FMS som användes i projektet var 3.5 och lanserades 13.01.2009. FMS kräver licens men det finns en utvecklingsversion som har en begränsning på 10 samtidiga anslutningar och är gratis ifall man registrerar sig på Adobes webbsida. Denna användes vid utvecklingen av programmet (Adobe, 2010d).

FMS har många fördelar eftersom den är gjord för att användas med Flash-applikationer. Den har färdiga funktioner för att enkelt kunna kalla på funktioner som ligger på antingen servern eller klienten. Den kan dessutom spara objekt med värden som enkelt kan hämtas från klienterna ifall de har rättigheter till dem.

FMS v3.5 har ändå en nackdel, förutom att den kräver licens. Serverkoden som används kallas för ”Server-side ActionScript” och är inte ett så modernt programmeringsspråk. Koden på servern kan inte göras så avancerad men fungerar perfekt som förmedlare av spelturer.

FMS är också ett mycket effektivt verktyg vid testning, hittande av fel och övervakning av serverns kraftanvändning via administratorkonsolen (se Figur 6: Flash Media Servers).



Figur 6: Flash Media Servers administrator konsol

4 PROJEKTETS FÖRVERKLIGANDE

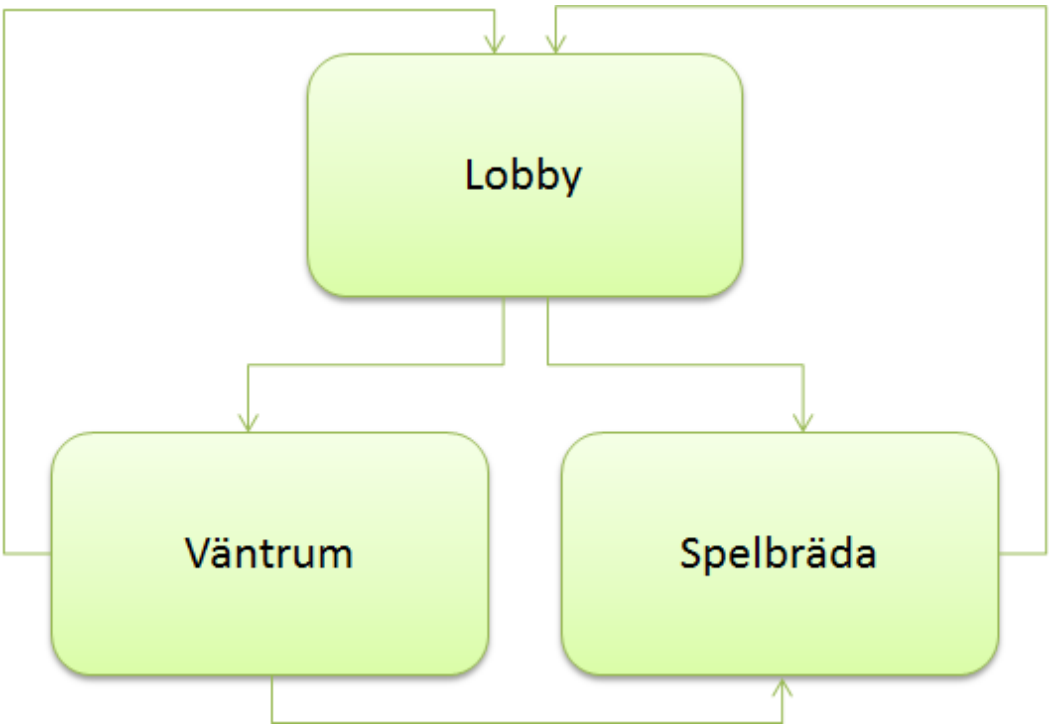
4.1 Introduktion till lösningarna

I detta kapitel beskrivs de lösningar som användes för att förverkliga projektet, med exempel, samt motiveringar till varför just dessa lösningar valdes. Användargränssnittet har varit en dominerande del av projektet. Av den orsaken fäster detta kapitel speciell uppmärksamhet vid hantering av massgrafikobjekt med hjälp av containers. All kod som behandlas i detta kapitel är AS3.

4.2 Uppbyggnad och flöde

Min uppgift i projektet var att programmera en klient till spelet. Det skulle finnas tre olika vyer; lobby, väntrum före spelet startas och själva spelbordet.

För att göra det lättare att förstå spelets flöde är den presenterad i sin allra enklaste form i Figur 7: Spelets flöde.



Figur 7: Spelets flöde

När spelaren tar kontakt med klientapplikationen hämtas spelarens unika inloggningsnamn från en session i bläddraren som sedan skickas till servern för att läggas till i den aktiva spelarlistan. När den nödvändiga informationen är skickad till servern kommer användaren in i lobbyn. Där syns andra spelare som befinner sig i lobbyn, spel som har sittplatser kvar och knappar för startande av ensamspelarläget eller för att starta ett eget spel med människospelare. Vid uppstart av eget spel kommer spelet sedan att synas för andra spelare i lobbyn. I lobbyn finns också knappar för att byta språk eller för att läsa reglerna.

I vår prototyp av programmet hade vi också en simpel chatt som spelarna kunde använda för att kommunicera med i lobbyn, men denna ville inte beställaren använda på grund av att den inte innehöll språkfilter.

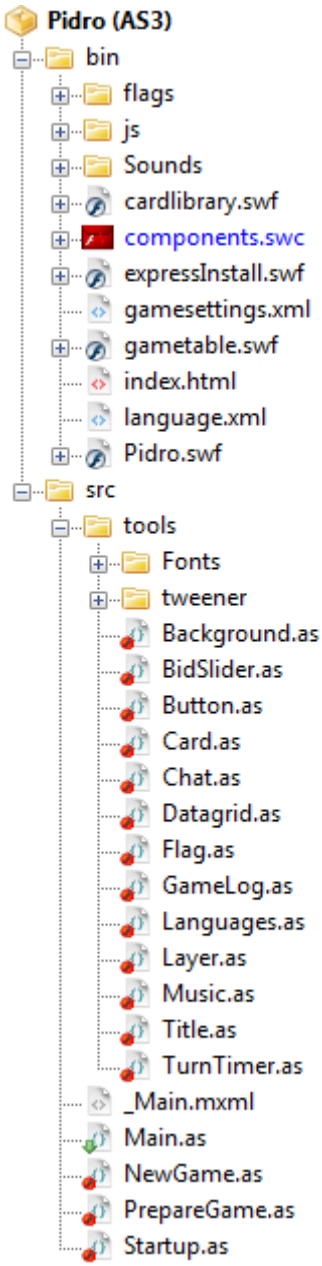
Från lobbyn har spelaren två val för att starta spelet; ensamspelarläge mot datorspelare eller ett spel med människospelare. Gör man valet att spela ensam tas spelaren direkt till spelbordet och spelet startas. Gör man valet att spela med flera människospelare kommer man att tas till ett väntrum där man kan välja lag (röd eller blå). I väntrummet har rummets värd alla rättigheter att starta spelet när han/hon vill, ifall alla spelare som

kommit in i spelet har valt sitt lag. I ett Pidro-spel behövs fyra spelare. Ifall kortspelet startas med tre eller färre spelare, kommer de tomma platserna automatiskt att ersättas med datorspelare. När värden har valt att starta spelet tas alla spelare till spelbordet, korten delas ut och spelet kan börja.

I spel med mer än en människospelare, kommer en klocka att räkna ner för varje spelares speltur. Hinner spelaren inte göra sitt drag under denna tid kommer spelaren att avlägsnas ur spelet och återföras till lobbyn. Varje gång en spelare blir utslängd eller lämnar spelet frivilligt, ersätts han/hon av en datorspelare.

När spelet är slut, kommer det ett meddelande till spelarna om vilket lag som vann och ger dem möjlighet att starta ett nytt spel.

För att få en klarare bild av programvaransuppbyggnad och strukturen kan man se på Figur 2: Struktur och uppbyggnad, som visar hela projektets filstruktur.



Figur 2: Struktur och uppbyggnad

4.2.1 Containers och dess användning

I en väldesignad Flash-applikation behövs en bra grund för grafikobjekten innan de läggs in. Detta sköts med så kallade ”containers”.

Enkelt förklarat är en ”container” någonting som man kan sätta grafikobjekt in i och senare lätt kontrollera dem därifrån. En ”container” är ett skärmbildsobjekt (DisplayObject) som kan vara t.ex. en bild, men oftast när man talar om en ”container” är det frågan om ett osynligt skärmbildsobjekt som innehåller andra skärmbildsobjekt

(Adobe, 2010e). En ”container” kan innehålla flera osynliga skärmbildsobjekt för vidare sortering eller synliga användargränssnittsobjekt. En ”container” används inte endast som en samlingsplats, utan t.ex. när det raderas, flyttas eller roteras så kommer allt som finns innanför den att följa med.

Låt oss säga att vi har en korg som innehåller några bröd, flyttar man på korgen så flyttar man också på bröden på grund av att bröden är placerade i korgen. Korgen representerar i detta skede ett skärmbildsobjekt som används som en ”container” för brödobjekten. Detta är ett exempel med ett lager av ”containers”. För att göra det litet mera invecklat så kan man lägga t.ex. ost på brödet, som då i sin tur är placerat in i ”brödskärmbildsobjektet” för att osten skall följa med brödet och hålla sin exakta position i jämförelse med brödet när brödet flyttas. En ”container” måste inte nödvändigtvis vara osynlig. Detta blir ett resultat av en korg som innehåller bröd som innehåller ost. Simpel kod för detta skulle se ut så här:

```
var korg:Sprite = new Sprite(); // Sprite är en typ av skärmbildsobjekt i AS3.
var brod: Sprite = new Sprite();
var ost: Sprite = new Sprite();
// I detta skede är våra skärmbildsobjekt skapade men ingenting syns på skärmen ännu.
this.addChild(korg); // Korgen på scenen, this = bottenlagret
korg.addChild(brod); // Brödet läggs i korgen
brod.addChild(ost); // Osten läggs på brödet
```

För att lägga till ett skärmbildsobjekt i Flash och få den synlig på den så kallade scenen (stage) måste klassen vara en förlängning (extends) av antingen klassen MovieClip eller Sprite.

```
public class ExempelKlass extends Sprite {...}
```

För att göra det lättare att hålla reda på z-indexen i ”containers”, det vill säga i vilken ordning objekten är ovanpå varandra, kan man använda sig av funktionen addChildAt(objekt, position). Ett bra exempel (se Figur 3: Containerhantering och z-indexering, och dess kod) på användning av detta är då man vill placera ett objekt längst bakom alla andra och flytta de andra objekten ett z-index framåt. I vanliga fall kommer alltid det nya skärmbildsobjektet överst.

```
var container:Sprite = new Sprite();
var cirkel1:Sprite = new Sprite();
var cirkel2:Sprite = new Sprite();
... // Här görs båds cirklarna till cirklar.
container.addChild(cirkel1);
container.addChildAt(cirkel2, 0);
```

Figur 3: Containerhantering och z-indexering

När cirkel1 läggs till med addChild() kommer dess z-index att vara 0. Men när cirkel2 sätts in på nollans position kommer cirkel1 att flytta z-index till 1. Detta kan bevisas med:

```
trace(container.getChildAt(0) == cirkel2); // skriver ut ”true”
trace(container.getChildAt(1) == cirkel1); // skriver ut ”true”
```

4.2.1.1 Varför containers är så viktiga

Containeranvändningen har många fördelar. Ordning och reda är speciellt viktigt i Flash projekt med många visuella objekt som skall röra på sig och om Flash resolutionen skall kunna ändras är det ännu viktigare för att allting skall hålla sin absoluta position. Utan en väl planerad containerstruktur kommer det troligen att leda till att man tappar greppet om alla skärmbildsobjekt. För att detta skall förstås bättre kan man tänka sig att man vill centrera flera spelkort i mitten av scenen. Skulle man bygga upp detta utan en container för korten, som färdigt är centrerade, skulle man vara tvungen att centrera varje enskilda kort.

Inte bara nog med att en kort container skulle lätt kunna hålla ordning på 52 kort, så kan föräldercontainern också användas för att ”loopa” igenom alla sina så kallade barn (child). Detta betyder att man kan använda containers också som en samlingsplats för dess barnobjekt. Som redan nämnts tidigare, om man tar bort föräldern med removeChild(), gömmer, ändrar genomskinligheten eller gör vilka som helst ändringar

på den så kommer det också att ändra alla barnobjekten. Barnet kan däremot inte påverka föräldern (Adobe, 2010f).

Andra allmänna, mycket användbara funktioner som kan användas på ett containerobjekt för att hitta och manipulera dess barn är:

contains(DisplayObject)

Returnerar en boolean med information om containern innehåller ett visst barnobjekt.

getChildByName(String)

Returnerar ett objekt från containerns barn med namnet man skrev som stringparameter.

getChildIndex(DisplayObject)

Returnerar en integer med informationen om displayobjektets z-indexposition.

setChildIndex(DisplayObject, int)

Byter z-indexpositionen för ett visst barn.

swapChildren(DisplayObject, DisplayObject)

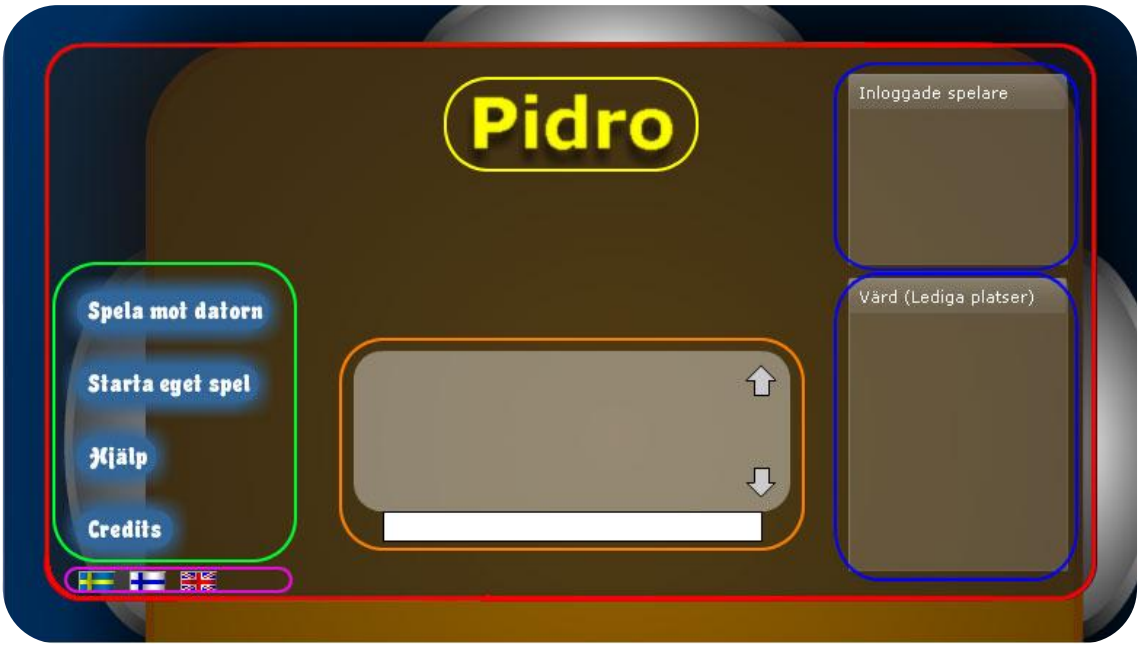
Tar emot två stycken displayobjekt och byter dess z-indexpositioner med varandra.

swapChildrenAt(int, int)

Byter z-index på två stycken barnobjekt, som söktes fram med dess z-index.

4.2.1.2 Användning av containers i projektet

Varje komponent eller komponentgrupp utnyttjar sig av containers, ingenting ligger fristående från allt annat. Spelets lobby är uppbyggd under ett huvudlager som innehåller datagridarna, knapparna, språkvalscontainern, titel och chat componenten (se Figur 4: Containerhantering för lobbyn).



Figur 4: Containerhantering för lobbyn

Huvudcontainer

Knappcontainer

Rubrik

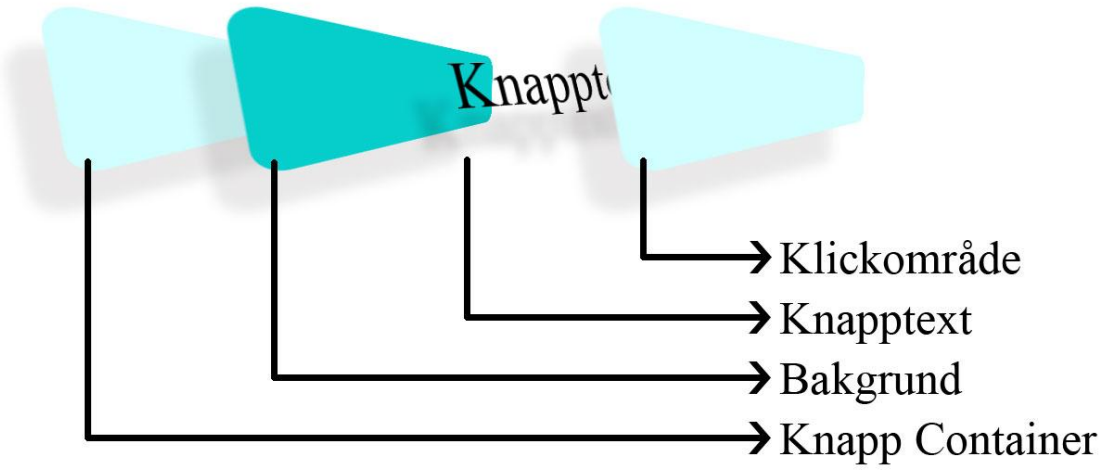
Chat

Datagrid

Dessa komponenter är lätta att placera in på skärmen på rätt plats inom huvudlagret. Också då man tar sig vidare från lobbyn försvinner hela huvudlagret och nästa vy laddas in. Borttagandet av alla komponenter som ligger i lobbyn sköts med en enda rad kod och kan fås tillbaka lika lätt. Det är väldigt lätt med hjälp av detta att göra animationer då man byter vy i spelet.

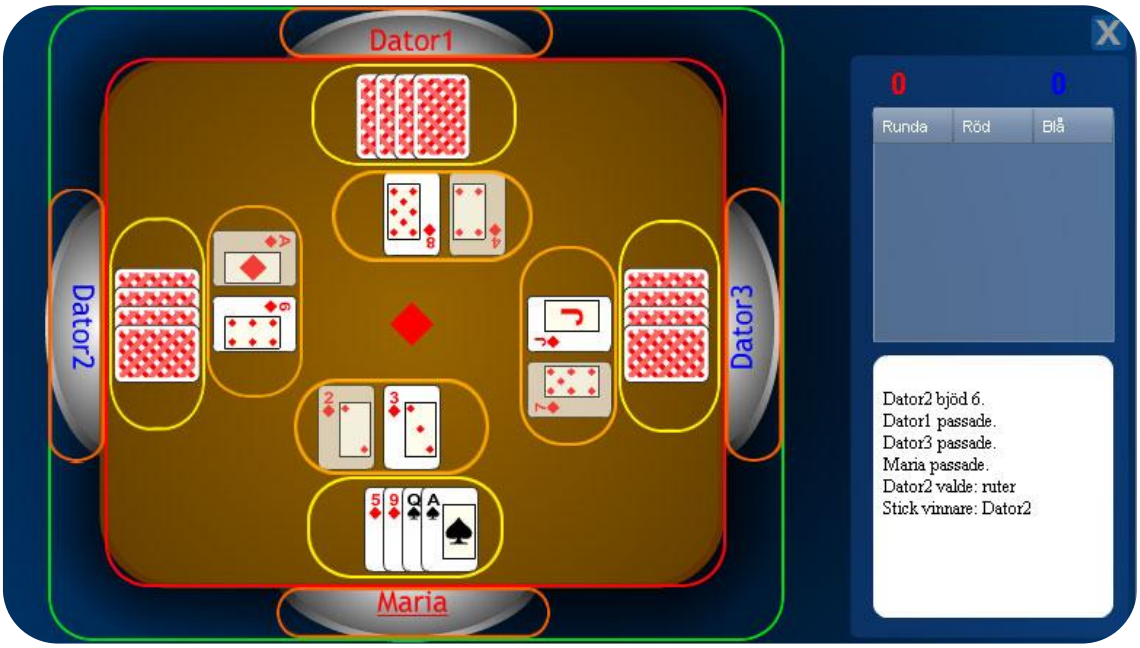
Containers är också mycket nyttiga vid uppbyggandet av knappar, där containers också har utnyttjats. Knapparna i projektet är designade att ha fyra olika lager. Dessa fyra lager är knappcontainern, bakgrund, knapptexten och ett osynligt klickområdslager som används för mustriggningar. Understa lagret, det vill säga knappcontainern, håller ihop hela paketet. Bakgrunden används naturligtvis som bakgrund och text som knapptexten. Klickområdslagret är viktigt för att man inte skall behöva göra en onödig mustriggare. Ifall den genomskinliga klickområdslagret (som är det lager som ligger högst uppe och fyller hela knappens område) inte skulle existera, skulle man ha muskontakt med både

knapptexten och bakgrunden. Detta skulle leda till att man antingen skulle vara tvungen att göra dubbla mustriggare eller att knappen bara skulle fungerar då man antingen har muspekaren rakt på texten eller på bakgrunden. Klickområdet triggas inte bara av en mus-klickning, utan också när muspekaren förs över eller tas bort från området. Vid ”MouseOver()” och ”MouseOut()” händelsefunktionerna körs animationer som ändrar på bakgrundslagrets färg.



Figur 5: Container- och lagerhantering för en knapp

Själva spelbrädan har också en liknande struktur fastän här används det två olika versioner av huvudcontainers. Ena huvudcontainern skulle man kunna kalla för spelområdscontainer medan den andra då skulle vara spelbordscontainer.



Figur 6: Containerhantering för spelbrädet

Spelområdescontainer

Spelbordscontainer

Kort som spelaren har på handen och osynliga för de andra spelarna

Kort som placeras eller har placerats tidigare på bordet

I Figur 6: Containerhantering för spelbrädet, ser man hur nästan alla element är centrerade på något sätt. För att göra detta är det mycket användbart att använda sig av föräldralagrets bredd och höjd. T.ex. sorten som det spelas i (i Figur 6: Containerhantering för spelbrädet) är ruter och är placerat i mitten av spelbordscontainern. Denna position räknas ut med:

X-position: (spelbordscontainerns bredd/2) - (sort-bildens bredd/2)

Y-position: (spelbordscontainerns höjd/2) + (sort-bildens höjd/2)

4.3 Kortleken

4.3.1 Kortlekens presentation

Då ett spelkort skapas, skapas det två eller tre olika lager beroende på hur kortet är vänt; en som en osynlig container, en för spelkortets fram- eller baksidas bakgrund och en för själva innehållet av kortet med sort och valör. Kort som är vända med framsidan uppåt innehåller alltid tre lager och har en vit bakgrund som placeras in i den osynliga containern medan kort som är nedåt vända bara har två lager; den osynliga containern och kortets baksida. Det tredje lagret på ett kort som är uppåt vänt innehåller kortets sort och valör, som är placerat in i bakgrundslagret. Detta gjorde det mycket lätt att justera kortets innehåll exakt på rätt ställe för att kunna se korten på ett klart och tydligt sätt också då de är placerade på varandra. För att försäkra att positionerna på korten alltid är exakta och att alla kort använder samma koordinater så placeras ännu kortcontainer in i spelbordscontainern.

4.3.2 Kortlekens uppbyggnad

Kortlekens containerstruktur, som beskrevs i föregående avsnittet ”Kortlekens presentation” har en litet speciell inladdningsmekanism som är mycket effektiv vid tillfällen där det används stora mängder grafik eller olika komponenter. Detta sköts via en .swc-fil (Adobe, 2010g). En .swc-fil är en slags förrådsfil, ungefär som .zip- och .rar-filer. Swc-filen hänvisas ofta till som ett klassbibliotek och innehåller flera olika filer, varav det viktigaste är själva .swf-filen och en catalog.xml-fil. Xml-filen håller reda på innehållet av komponentpaketen och de enskilda komponenterna i filen (Adobe, 2010h).

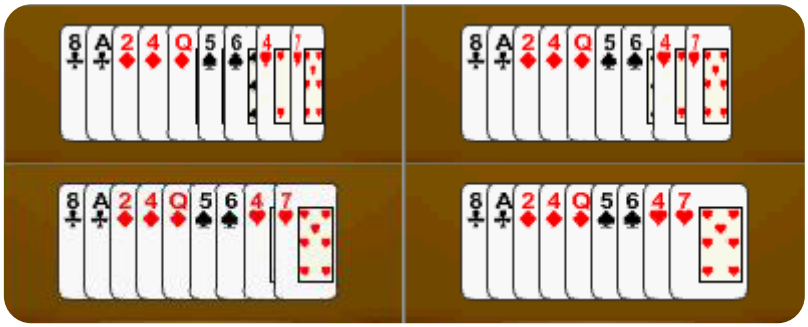
När kortleken skapades gjordes den med Adobe Flash som ett helt normalt Flash projekt. Alla komponenter som behövdes för korten (kortets bak- och framsida, spader (s1-s13), klöver (c1-c13), ruter (d1-d13) och hjärter (h1-h13)) skapades som egna objekt och sparades som symboler i Flash-filens bibliotek. Sedan exporterades filen till en .swc-fil och alla korten kunde laddas in via AS 3 till projektet. För att få fram ett komponentpaket kan man importera paketet till projektet, men för att få tag på andra objekt som t.ex. bilder, ljud och animation måste man veta objektets namn i biblioteket.

Swc-filerna är inte bara nyttiga för att med en enda laddning kunna ladda in stora mängder data med en välorganiserad struktur, utan den blir också mycket användbar i situationer där man vill ha möjlighet att enkelt byta ut komponenterna. Ett bra exempel är att man skulle skapa flera olika kortlekar med samma namnstruktur och med hjälp av detta kunna byta kortlek ”on-the-fly” eller låta användaren välja vilken kortlek han/hon vill använda.

4.3.3 Utdelning av kort

Då själva spelkorten som spelaren kommer att spela med skapas, skapas de nio korten servern bestämt osynligt redan på sin rätta plats före den själva kortutdelningsanimationen händer. Då utdelningen av korten skall börja skapas det en kortlek med bara baksidor av nio kort för varje spelare, det vill säga sammanlagt 36 kort. Av dessa kort kommer bara medspelarnas kortbaksidor att användas efter att spelet har börjat. Efter att kortleken är skapad startas en timer som kör en funktion 36 gånger (för varje kort) med ett visst hastighetsintervall. Funktionen som triggas gör en kortutdelningsanimation från kortpacken i mitten till de fyra olika spelarna. Korten delas ut turvis och utdelningen börjar alltid från spelare1. För varje kort som delas ut håller en variabel reda på till vilken spelare nästa kort skall gå. När alla korts baksidor är utdelade startas en annan timer som kör igång en funktion som körs endast på spelarens kort, ett i gången, med intervallet 0,1 sekunder. Funktionen som timern kör igång används för att skapa en animation som ser ut som om spelarens kort skulle vändas. De tre andra spelarnas kort förblir nedåt vända. Kortbakgrunderna som är skapade och som var använda för att animera utdelningen av korten har ingen framsida och dessa kort har heller inga värden som säger vilka kort de är. Kortvändningsanimationen för spelaren som skall se sina egna kort animerar bredden på kortbakgrunderna till noll för varje kort från höger till vänster. Hastigheten som korten animerar bredden till noll körs med 0,2 sekunder. Detta betyder att kortet blir för stunden helt och hållet osynligt. Direkt när en av kortbakgruderna har fått bredden noll kommer den gömda framsidan av kortet som kortbakgrunden representerar att börja animera från bredden noll till kortets fulla bredd med igen samma 0,2 sekunders hastighet för att det inte skall märkas att det är en ny animationsfunktion. Detta skapar en visuell illusion och det ser ut som om kortbakgrunden faktiskt skulle ha vänt sig.

Eftersom varje kort börjar animera med hastigheten 0,1 sekunder efter varandra och animationen för att vända kortet till bredden noll och tillbaka tar båda 0,2 sekunder kommer resultatet att se ut som en snygg våg-animation av kortvändningen. Efter att korten är klart animerade tas de onödiga kortbakgrunderna som fortfarande ligger osynliga på skärmen bort. Som redan tidigare sagts har kortbakgrunderna för de andra spelarna inga värden och klienten kan därmed omöjligt få reda på vilka kort motståndaren har, vilket betyder att fusk inte kan ske.



Figur 7: Kortutdelning - visuell illusion

4.4 Effektivisering av spel och spelbräda

För att ett nätverksspel skall fungera och i detta fall då det gäller upp till fyra spelare i ett spel, är det viktigt att också tänka på den enskilda spelarens perspektiv över spelet. Väntetider är en av de viktigaste sakerna att ta i beaktande men också saker som hur lättförestående spelets framskridande är för en nybörjare viktigt med tanke på att spelaren skall komma tillbaka och spela på nytt. Om en spelare blir missnöjd eller trött på spelet kommer han/hon troligen att lämna spelet som resulterar i att hans/hennes plats tas över av en datorspelare och en datorspelare tycker de andra människospelarna knappast lika mycket om att spela med som den mänskliga.

Spelbrädan skall vara tydlig och lätt att förstå, laddningstiderna skall vara minimala och små inställningar på spelet skall kunna göras utan att programmera om och kompilera programmet igen. I ett kortspel är det naturligtvis kortleken som är den viktigaste delen att bygga upp och presentera tydligt på spelbordet, men det är även viktigt att tydligt se vem ens lagkamrat är, vilka kort som är spelade och poängen.

4.4.1 Spelinställningar

Ett flexibelt spel innehåller ofta en konfigurationsfil för spelinställningar. För detta projekt skapades en xml-fil med inställningsvariabler som administrativa personer kan ändra på medan spelet är i användning utan att det vållar några problem. I riktigt stora spel är en konfigurationsfil väldigt viktig och är ofta också tillgänglig för användaren att ändra på små saker, oftast i grafiken, för att göra spelet mera passande för den dator de använder. I ett Flash-projekt som detta, behövdes ingen användarkonfigurationsfil och Flash har en inbyggd funktion för inställning av tre olika grader av grafikkvalitet.

Inställningsfilens uppgift är närmast att innehålla ett numeriskt värde för spelturstiden i sekunder, det vill säga hur länge en person har tid att tänka under spelets gång förrän han/hon avlägsnas från spelet och ersätts av en datorspelare. Filen innehåller dock andra små inte så viktiga inställningar som t.ex. kortutdelningshastighet och hur snabbt korten skall vändas.

4.4.2 Time-out för spelare

Varje gång det blir en spelares tur startas det en timer i klienten uppe till vänster som efter ett visst antal sekunder avlägsnar spelaren från spelet till lobbyn. Spelaren blir då direkt ersatt av en datorspelare för att spelet skall kunna gå vidare. Time-outen kan ställas ”on-the-fly” i spelets xml-fil för inställningar till vilken tid som helst men en tid runt 30 sekunder rekommenderas.

Time-out har sina för- och nackdelar. Ibland kanske man skulle vilja ha litet mera tid att tänka andra gånger är valet lättare att göra och är då också mindre tidskrävande. Dock kan man inte ha ett dylikt spel utan en timeout på grund av att hela spelet inte skulle gå vidare ifall en spelare slutar spela utan att lämna spelet. Time-out funktionen används endast då en spelare spelar mot andra människospelare. Då en spelare spelar ensam med 3 datorspelare aktiveras inte time-out funktionen.

4.5 Flexibilitet för vidareutveckling

Ett stort programmeringsprojekt kräver god planering. Man måste veta hur man skall göra små enskilda komponenter för att det skall kunna fungera på ett bra sätt ihop med

andra komponenter och med hela projektet. För att någonsin bli klar med sitt projekt är man också tvungen att gallra ut mycket funktionalitet som man kan flytta till vidareutveckling.

4.5.1 Planering och struktur

Projektets programkod är fullständigt objektorienterat och alla element har sin egen klass. Med hjälp av en mycket väl planerad struktur från första början, fick projektet en klar och tydlig uppbyggnad med kod som är lätt att förstå. För att uppbyggnaden har tvingat nästan allt till egna klasser, är själva uppbyggnadsklasserna, som sätter ihop hela projektet, mycket strukturerade på grund av att man med endast några rader kod kan skapa objekten, sätta dess parametrar och visa dem på skärmen. Här under följer ett exempel på hur man skapar en knapp med objektorienterad kod.

```
var btn1:Button = new Button(”Tryck på mig”);
buttonContainer.addChild(btn1);
```

4.5.2 Språkval

Ett av kraven för projektet var att kunna skapa, ändra och ta bort språk i spelet på ett lätt sätt. Inga hårdkodade texter fick existera i koden. För att göra detta fanns det egentligen två bra alternativ, databas eller xml. Med tanke på att databaser inte skulle komma att används i projektet för något annat än språkvalen, var xml ett lätt val. Ifall databaser skulle ha använts skulle man gärna ha en administrationswebbsida för språk där man skulle kunna editera, ta bort och skapa nya språk.

Lösningen för att hantera språkval löstes i projektet därmed med xml. Xml-filens yttersta block som är <Languages> innehåller <Language> block där språköversättningarna finns. När spelet startas laddas xml-filen in i programmet och kollar vilket <Language> block som har parametern "default=true". Detta språk kommer att bli det språk som programmet startas med. Varje som kan bli synlig på skärmen finns i denna xml-fil. När man vill skapa ett nytt språk skapar man ett nytt <Language> block innanför <Languages> blocket och översätter alla innevarande block till det nya språket. När programmet märker ett nytt språkblock kommer den automatiskt vid nästa uppstart att rita en ny flagga för det nya språket i

språkvalscontainern (se Figur 8: De olika språkens representativa flagga laddas in enligt hur många språk som är skapade.). Flaggbildens länkning kommer från blocket <Image> som har namnet på flaggbildsfilen som är placerad i mappen ”flags”.

Detta dynamiska språkvalshanteringssätt har visat sig fungera utmärkt. En administrator kan i vilket skede som helst ändra på alla texter i hela spelet utan att användningen av spelet störs.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Languages>
        <Language default=”true”>
                <Lang>Swedish</Lang>
                <Image>sweden.gif</Image>
                <Title>Pidro</Title>
                <NewGameBtn>Spela mot datorn</NewGameBtn>
                .....
        <Language>
                <Lang>Finnish</Lang>
                .....
        </Language>
        <Language>
                <Lang>English</Lang>
                .....
        </Language>
</Languages>
```

Figur 8: De olika språkens representativa flagga laddas in enligt hur många språk som är skapade.

Då programmet startas laddas .xml-filen in i Flash och det skapas ett xml objekt av data som laddats in. Från xml-objektet skapas det en XMLList för det aktiva språket. Det språket administratorn har valt att skall vara standard har ett attribut ”default = true” på sitt language-block. Detta språk laddas in automatiskt till XMLList-objektet med:

```
var xmlLang:XMLList = xmlData.Language.(attribute(”default”) == ”true”);
```

Ifall man byter språk genom att klicka på en av flaggikonerna kommer en funktion att triggas som ändrar XMLList-objektet enligt:

```
var xmlLang:XMLList = xmlData.Language.(Lang == e.currentTarget.language);
```

e.currentTarget.language är en länkning till den specifika flagobjekets språksträng som berättar vilket språk det är frågan om.

För att använda sig av själva språket använder man sig av XMLList objektet enligt följande exempel:

```
var newGameBtn:Button;
newGameBtn = new Button(xmlLang.NewGameBtn);
```

Denna kod skapar ett nytt Button-objekt och skickar xmlLang.NewGameBtn-strängen till dess konstruktör. NewGameBtn som är ett block i varje språk i xml-filen och som används som texten för knappen kommer från xmlLang-objektet som alltså är en XMLList. Byter XMLList information körs det bara en uppdateringsfunktion för texterna.

4.6 Nätverkanslutning

Spelets huvudsakliga idé är inte att kunna spela det ensam, även om den också har funktionalitet att spela mot datorspelare. Ett kortspel där man spelar två mot två blir mycket intressantare ifall man har mänskliga medspelare. För att detta skall lyckas krävs det en nätverksanslutning av något slag. Det finns olika sorter av lösningar för situationer som denna, men för det här spelet där det är frågan om upp till fyra spelare per bord och ett visst antal spel som körs på samma gång kommer inte simpla lösningar, så som att skicka data mellan klienterna att fungera tillräckligt bra. I dessa fall skulle också säkerheten och datorspelarnas kontakter ställa till med stora problem.

För att lösa problemet med nätverksanslutningar togs Adobe Flash Media Server (FMS) i bruk efter mycket testande av olika lösningar. FMS var en bra lösning för att få en ordentlig struktur på spel, spelare och all annan information som behövs sparas på servern. För vidare utveckling är FMS också ett bra verktyg för att Adobe utvecklar den konstant. I användning av FMS kan man dela upp koden bra mellan serverkod och kli-

entkod, så all kod som inte behöver vara på klienten hålls på servern. Servern kan ensam hålla reda på turer och dela ut uppgifter åt klienterna som berättar när de är klara eller om någonting har hänt. Inga konstanta loopar eller massvis med eventtriggare behövs heller på grund av att man med FMS kan använda sig av delade objekt och synkoniseringsfunktioner (Adobe, 2009).

4.6.1 Kontakt till mediaservern

För att kontakta FMS med en Flash-klient måste man göra upp en så kallad ”NetConnection-klass”. Från NetConnection-klassen kopplar man sig till FMS genom ett protokoll som heter Real Time Messaging Protocol (RTMP). RTMP är utvecklat av Adobe och används för att flytta data över internet mellan klient och server (Wikipedia, 2010c). RTMP baserar sig på TCP-protokollet och använder sig av port 1935.

En kontakt till FMS görs enligt:

```
nc.connect(“rtmp://localhost/alpha”, playerName);
```

Nc som är NetConnection-objektet kallar på funktionen ”connect” som tar första parametern som en adress till vart NetConnectionen skall koppla upp sig. För denna parameter, när det är frågan om att koppla upp sig till en FMS så används RTMP-protokollet som följer med serverns IP och namnet på serverns applikation.

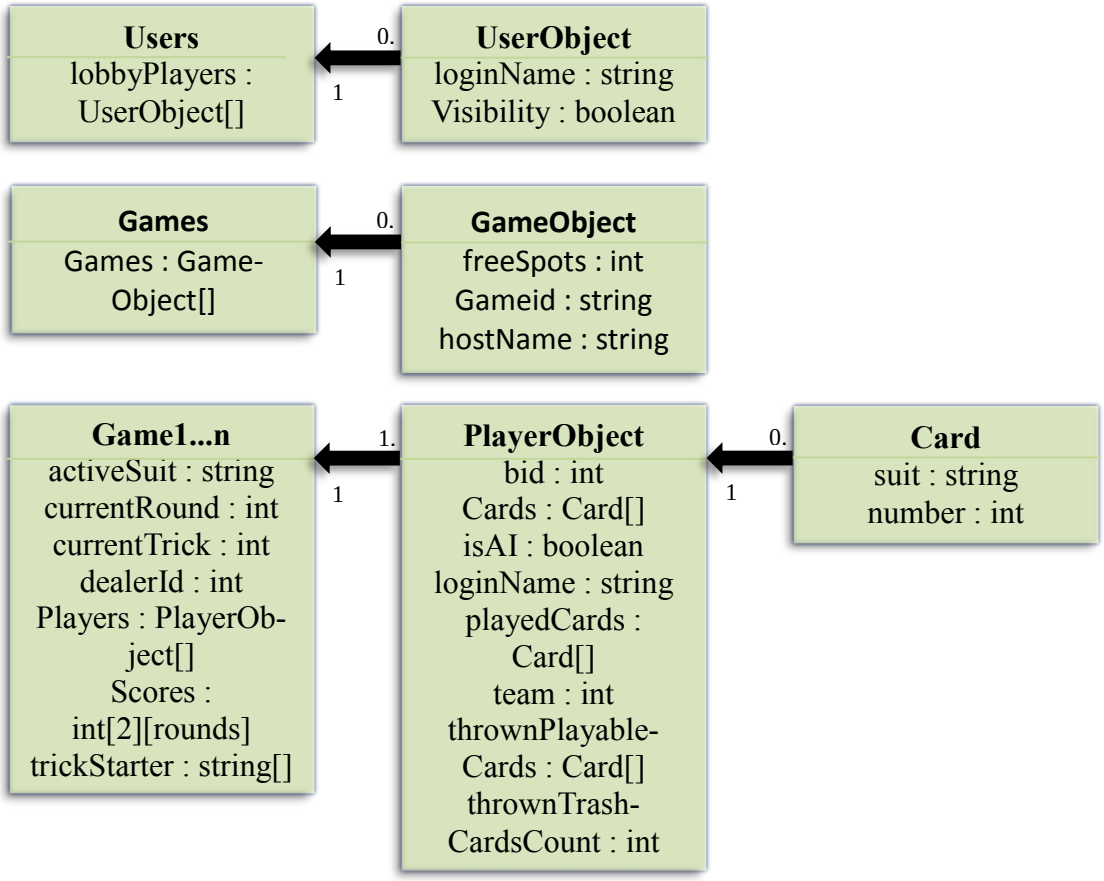
För att sedan kolla om man alls har fått kontakt och så man vet när man kan börja använda sig av FMSs funktioner startar man en händelselyssnare (event) som i nästa exempel:

```
nc.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
```

I netStatusHandler funktionen kan man sedan kolla med händelseobjektet ifall kontakten lyckades. Har kontakten lyckats är klienten klar att börja användningen av FMS funktionalitet.

4.6.2 Delade objekt

Delade objekt (shared objects) är objekt som ligger på servern (Adobe, 2010i). Dessa objekt kan flera klienter och servern ha åtkomst och använda samtidigt. Både servern och klienten kan skapa och uppdatera dessa objekt. Ifall man inte behöver spara data man vill sända över nätet, kan man använda delade objektets ”send” metod. Bra exempel för att använda send-metoden är för en chatt eller för en spellogg som används i detta projekt. För projekt användes också en hel del normala delade objekt. Figur 9: Projektets delade objekt och dess relationer, visar vilka delade objekt som sparas på servern och används av både klienterna och servern.



Figur 9: Projektets delade objekt och dess relationer

Users, innehåller en tabell med *UsersObject* (användarobjekt) för varje användare. Användarobjekten innehåller en sträng för namnet och en boolean som används för att kolla om användaren skall synas i lobbylistan.

Games, innehåller motsvarande data som Användarobjektet, men här är det en tabell med *GameObject*(spelobjekt). Ett spelobjekt används för att hålla reda på hur många lediga platser det finns kvar, spelets ID och värdens namn. Denna information syns i lobbyn för spel som går att gå med i.

Game1...n, innehåller all information som ett egentligt spel behöver och det skapas när ett spel startas. Den håller reda på i vilket sort man spelar i, vilken runda och vilket stick som pågår. Objektet håller också informationen om vem som är utdelaren av korten och vem som påbörjade sticket. En tvådimensionell tabell håller reda på båda lagens poäng och spelarobjekten sparas även i en egen tabell i objektet. För varje spel som skapas, ökar hela tiden objektens namn från Game1...n-antal spel som är igång.

PlayerObject, ett spelobjekt innehåller flera spelarobjekt. Spelarobjektet håller reda på allt man behöver veta om en spelare t.ex. är det en datorspelare, hur mycket spelaren har bjudit, vilket lag spelaren hör till, vilka kort har spelats och vilka kort som ännu ligger i handen.

Card, som står för kort har i sin tur en 1 → många relation till spelarobjektet och innehåller spelkortets sort och valör.

4.6.3 Synchandler

På varje delat objekt kan man skapa en så kallad ”synchandler”. Synchandlern är en syncevent som triggas ifall ändringar sker i det delade objektet (Adobe, 2010j).

```
gamesSo.addEventListener(SyncEvent.SYNC, gamesSyncHandler);
...
private function gamesSyncHandler(e:SyncEvent):void {...}
```

4.6.4 Servercalls och spelturer

Istället för att sköta all kontakt till FMS via delade objekt, är det också möjligt för både servern och klienten att köra igång funktioner på varandra via RPC (Remote Procedure Call). Detta är mycket användbart i hantering av spelturer.

När ett nytt spel körs igång gör servern ingenting före alla fyra spelare har gjort en servercall och meddelar att all information på rutan har laddats in och korten är utdelade på deras klient.

```
nc.call(“readyToBid”, rs, gameCode, playerName);
```

Efter att alla fyra klienter har kallat på serverfunktionen ”readyToBid” med den unika spelIDn och spelarnamnet, som också alltid är unikt enligt kraven, vet servern att alla spelare är klara och spelet kan köras igång.

Variabeln ”nc” som kallar på funktionen ”call” är en NetConnection-klass. NetConnection-klassen används, som tidigare förklarat, för att koppla upp sig till FMS och skapa delade objekt. Variabeln ”rs” är en Responder-klass som kan användas med NetConnection.call-metoden. Responderns konstruktör tar emot en variabel som är ett funktionsnamn.

```
private var rs:Responder = new Responder(onReply);
...
private function onReply(result:Object):void
{...}
```

Efter att en NetConnection.call är gjord kan den som tagit emot RPCn köra en respons tillbaka med ett objekt som parameter. Detta är dock inte nödvändigt att använda sig av.

När spelet väl körts igång kallar servern på klienterna i tur och ordning och frågar hur mycket de vill bjuda, buden skickas tillbaka till servern och så vidare.

Spelet använder sig även mycket av RPC i själva lobbyn, som t.ex. när en spelare skapar ett spel, går med i ett spel, skriver i chatten, väljer ett lag, lämnar spelet eller startar ett spel.

5 DISKUSSION

Möjligheterna och alternativen för lösningar är många när det är frågan om programutveckling. Flash som var satt som krav av Rundradion för projektet kunde inte påverkas men kontakten mellan servern och klienterna var helt öppen. Flash med FMS visade sig fungera mycket bra som motor för även större Flash-nätverksspel i jämförelse med andra lösningar som testades. FMS kunde snabbt och enkelt förmedla information till klienterna och klienterna kunde läsa från delade objekt på servern och även köra vissa funktioner på servern själv. I jämförelse ifall man skulle ha gjort projektets serverkod med t.ex. PHP, har FMS den fördelen att den kan tillfälligt lagra information som är lätt att synkronisera med klienterna med en säkerhet. PHP med en databas skulle vara en annan säker lösning men då skulle programmet ha blivit tungt och långsamt på grund av de konstanta databasförfrågningarna för att hålla alla spelare synkroniserade med spelets framskridning.

Utvecklingen av själva kortspelet Pidro blev klart inom utsatt tid och avek mycket lite från den gjorda kravspecifikationen.

Fastän klientapplikationsplattformen var satt som krav för projektet är jag mycket nöjd över hur Flash klarade av detta och skulle inte i framtiden använda någon annan platform för liknande applikationer.

KÄLLOR

Adobe. 2010a. Adobe Flash Player [www]. Hämtat 13.4.2010. http://www.adobe.com/products/flashplayer/

Adobe. 2010b. Adobe Flash Player penetration [www]. Hämtat 13.4.2010. http://www.adobe.com/products/player_census/flashplayer/

Adobe. 2010c. Adobe Flex [www]. Hämtat 13.4.2010. http://www.adobe.com/products/flex/

Adobe. 2010d. Adobe Debuts Flash Media Server 3 Product Line [www]. Hämtat 14.4.2010. http://www.adobe.com/aboutadobe/pressroom/pressreleases/200712/120407adobefms3.html

Adobe. 2010e. DisplayObject [www]. Hämtat 26.4.2010. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/DisplayObject.html

Adobe. 2010f. Working with display object containers [www]. Hämtat 26.4.2010. http://help.adobe.com/en_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7e36.html

Adobe. 2010g. SWF files [www]. Hämtat 3.5.2010. http://livedocs.adobe.com/flex/3/html/help.html?content=building_overview_5.html

Adobe. 2010h. About SWC files [www]. Hämtat 3.5.2010. http://livedocs.adobe.com/flex/3/html/help.html?content=compilers_30.html#135468

Adobe. 2009. Beginner's guide to streaming video with Flash Media Server 3.5 [www]. Hämtat 14.4.2010. http://www.adobe.com/devnet/flashmediaserver/articles/beginner_vod_fm3.html

Adobe. 2010i. Understanding shared objects [www]. Hämtat 19.5.2010. http://www.adobe.com/livedocs/fms/2/docs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000049.html

Adobe. 2010j. Shared objects [www]. Hämtat 19.5.2010. http://help.adobe.com/en_US/FlashMediaServer/3.5_Deving/WS5b3ccc516d4fbf351e63e3d11a0773d37a-7fff.html

Flashdevelop. 2010a. FlashDevelop is a free and open source code editor [www]. Hämtat 13.4.2010. http://www.Flashdevelop.org/wikidocs/

Flashdevelop. 2010b. Flashdevelop community [www]. Hämtat 13.4.2010. http://www.flashdevelop.org/community/

Greensock. 2010a. Greensock hemsida [www]. Hämtat 14.4.2010. http://www.greensock.com/

Greensock. 2010b. TweenLite – A Lightweight, FAST Tweening Engine. Hämtat 14.4.2010. http://www.greensock.com/tweenlite/

Mantis. 2010. Mantis hemsida [www]. Hämtat 13.4.2010. http://www.mantisbt.org/

Pidro Online. 2011. Pidro [www]. Hämtat 3.5.2011. http://pidro.net/

Wikipedia. 2010a. Adobe Flash [www]. Hämtat 15.4.2010. http://en.wikipedia.org/wiki/Adobe_Flash

Wikipedia. 2010b. Apache Subversion [www]. Hämtat 15.4.2010. http://en.wikipedia.org/wiki/Subversion_%28software%29

Wikipedia. 2010c. Real Time Messaging Protocol [www]. Hämtat 20.5.2010. http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol

BILAGOR

BILAGA 1

Pidro projekt för YLE (06.02.2009)

Regler och begrepp

- 4 spelare (2 mot 2)
- Spelvärd (host) den spelare/användare som startar ett nytt spel
- Datorspelare (AIn) heter DatorN (N=1,2,3)
- Färg eller sort är benämningen på den sort man spelar i, t.ex. hjärter, spader osv.
- Pidro är båda femmorna i den färg som spelas (Hjärter och ruter fem, alternativt spader och klöver 5).
- Poängkort är Ässet = 1p, Knekten = 1p, Tian = 1p, Tvåan = 1p, 2 st Pidro = 5p vardera i vald sort. Totalt alltså 14p.
- Max 14 poäng per omgång
- Spelbara kort = 14 kort (vald sort 13 st + samma färgs Pidro 1 st)
- Icke-spelbara kort = Alla förutom spelbara kort
- Varv eller stick är en del av en omgång när varje spelare sätter ett kort vardera i samma färg (om man inte är utan kort i färgen som spelas).
- Vinnare av hela spelet blir det lag som först erhållit eller överskridit 62 poäng. Om båda lagen går över 62 poäng under samma omgång vinner det lag som bjöd
- budgivning:
  om tre första buden är pass, bjuder kortgivaren minst 6
  om någon bjuder 14, vinner den budgivningen
- med 9 kort på hand:
  kastas 3 - 9 kort
  man får inte kasta spelbara kort, förutom om man har fått mera än 6 st av dem
  man får aldrig kasta poängkort
  man får behålla icke-spelbara kort
- sista spelaren (den som är tredje i turen) blir utan kort, ifall korten inte räcker till
- bortkastade och spelade ”spelbara” kort skall synas för alla spelare

Allmänt

- Utvecklingsomgivningen erbjuder maximalt 10 anslutningar till utvecklingsservern. Detta betyder maximalt på 2 simultana spel under utvecklingsskedet
- ”load balance” testning kan inte göras i utvecklingsskedet
- Man måste vara inloggad på X3M's community för att kunna spela
- Man kan endast spela i ett pågående spel per loginnamn
- Användarnamn fås från cookie (exempel fås från YLE)
- Man kan inte hoppa in i ett spel som redan har börjat
- Man kan inte följa med andras spel (spectator)
- Om man blir disconnected under ett spel ersätts man av en datorspelare
- Spelaren meddelas med ljud då spelet kräver uppmärksamhet
- Resolutionen på Flash får vara max 780x440 (439) (relation 16:9)
- Inga hårdkodade texter
- Alla drag som sker efter att spelet har satt igång är tidsbegränsade

BILAGA 2

Kravspecifikation för spelmotorn (Krister Bäckman)

Lobbyn

player connects to the game

new player object is created

present GUI **(draw GUI)**

GUI consists of buttons for new game with 3 AIs, new game, list of games that have not started, help, and status (how many games are currently active, how many games have been played)

press button to start game with 3 AIs → start new game → jump to "spelets gång"

press new game → you start a new game and are shown the "new game screen"

press a game in the list → you are taken to the "new game screen" of that game

press help → show the game rules

Help screen

Show the rules and a back button **(if the rules don't fit on one page include buttons for viewing multiple pages)**

press the back button → you are taken back to the lobby

"New game screen"

In the new game screen there are four buttons corresponding to the seats in the table and a leave button. The host also has a start game button. The game gets the name of the host, the host name is also visible in the lobby

as long as there are seats free the game shows up in the lobby new game list

when you enter the "New game screen" you are automatically placed in a seat

the host can start the game with any ratio of human / AI players

press an empty seat button → you switch seats

press the leave button → you are taken back to the lobby

host presses the start button → jump to "spelets gång"

if there are free seats when the host presses start the seats are filled by AI players

Spelets gång

Initialization

set up game log **(draw log)**

set up players (AI / human) **(draw player icons)**

create deck **(draw deck)**

shuffle deck

deal from deck 9 cards per player, direction: left of dealer **(draws cards)**

dealer rotates left after every round **(draw dealer mark)**

the host is the first dealer

Betting

player left of dealer starts

everyone takes turns rotating around the table **(needs AI decision) (draw gui for selecting bet and draw everyone's bet)**

the cycle ends at the dealer

minimum bet is 6

max bet is 14

you have to bet higher or pass

if everyone else passes dealer bets minimum 6

the one with the highest bet chooses sort to play **(draw selected sort)**

players can now select which cards to throw away or keep **(needs AI decision) (draw cards and make selection possible)**

players can have max 6 cards in their hand

players then get cards from the deck until they have 6

if the cards run out they play with less than 6 cards

dealer gets to choose from all the leftover cards

if you have more than 6 playable cards you have to “kill” overflow cards

overflow cards are shown on the table

you cannot throw playable cards other than when you have more than 6 of them

you cannot throw point cards

Playing the game

Round 1

the highest bidder selects a card to play **(needs AI decision) (draw cards for human**

players)

players take turns selecting cards to play **(needs AI decision) (draw cards for human players)**

if a player runs out of cards to play he throws them on the table **(draw cards)**

points are shown all the time **(draw points)**

the highest card wins the round 2,3,4,...J,Q,K,A

when round is over bet is checked against points and points are given to the winning player pair **(draw scoreboard)**

players play consecutive rounds until 62 points

at 62 points the game ends

players are asked if they want to play again and if they all say yes a new game is created for them

Game over

clients are disconnected and directed back to the lobby (player object is reset)

AIs are freed from memory (AI destructor cleans up)

results are saved for statistics

Pidro på X3M commnunityn

- Skall finnas en API som möjliggör att man kan hämta information om status
- Lyfter info om hur många som finns i lobbyn
- Visar hur många spel som är igång

BILAGA 3

Pidro Datorspelare (AI) – Kravspecifikation (Peter Saloviin)

Följande utförs av datorspelaren:

- ger bud
- väljer sort, ifall högsta bud
- slänger kort
- under varv/stick: sätter ut kort

Den får inte veta de andra spelarnas kort, även om dess par är också AI. AI:n diskuterar inte sinsemellan för att förenkla deras spelande. AI:n tar inte i beaktan vem den spelar emot (användarnamn eller AI). AI:n väntar inte med sina beslut.

Det görs intervjuer med erfarna Pidro-spelare för att använda deras kunskap om spelet vid utveckling av spelstrategin. Litteratur om spelteori för Pidro, eller kortspel med liknande spelproblem, undersöks också.

BILAGA 4

Pidro Spelplan (Ron Holmström)

Klient delen

- Grafiken levereras från Yle inom två veckor (20.2.09)
- Fritt valbar uppbyggnad men med YLEs grafik
- Spelvy uppifrån, som t.ex. hearts spelet
- Poängräkning och logg till höger om spelbrädan, synlig hela tiden.
- Möjlighet att kunna ändra språk
- Möjlighet att en admin lätt kan skapa nya språk

Lobby

- En rubrik
- Lista med inloggade användare
- Lista med spel som ännu inte är startade
- Spel som är igång skall inte synnas i lobbyn, inte heller spelare som redan är inne i spel.
- Skapa nytt spel-knapp för att starta ett singelspel med 3 datorspelare
- Skapa nytt spel-knapp för att starta ett spel där andra mänskliga spelare kan gå med.
- Regler
- Credits

Väntrum

- Möjlighet för spelarna att välja lag, röd eller blå
- Värden kan starta spelet då alla spelare har valt lag
- Ifall alla sittplatser inte är tagna skall dessa lediga platser ersättas med datorspelare

Spel vy

- Skall man lyfta korten eller bara klicka?
- Ljud/musik för kort utdelning
- Varnings ljud för din tur att ge kort
- Ljuden levereras från YLE 20.2.2009

BILAGA 5

Vidareutveckling

- Individuell statistik
- Lösenord till spel
- Olika AIn
- Spara spelet
- Spela utan inloggning
- Chatt
- “Load balance”-testning