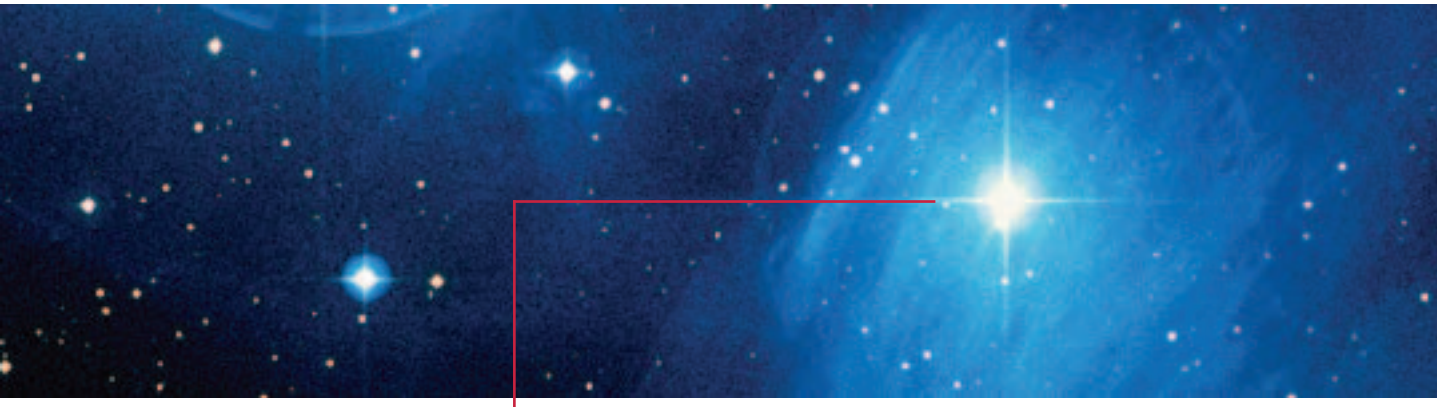
Mikael Bondestam Johan Isaksson

Spelprogrammering

med CDX och OpenGL

DOCENDO

Del 2



SPEL

HISTORISKT SPEL

Mål

Ett historiskt spel, då det är en av tidernas största datorspelssucceér och även det första (?) datorspelet du kommer att skapa. Men förhoppningsvis inte det sista...

Pong är med dagens mått mätt ett mycket enkelt spel, men det innehåller ändå några av de koncept som finns i mer avancerade spel. Ett antal av de första stegen på din långa resa för att bli spelprogrammerare kommer vi att gå igenom i detta kapitel nämligen hur man

- får någonting att röra sig på skärmen (bollen och senare plattorna),
- gör för att låta spelaren styra saker på skärmen med tangentbordet (plattorna),
- kan avgöra om saker på skärmen krockar med varandra (bollen, plattorna, väggarna),
- spelar upp ljud,
- håller reda på liv och spelets status.

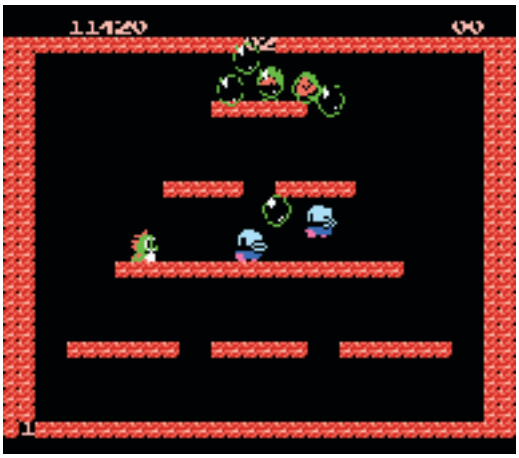
En gammal klassiker.

7. STUDSBOLL – EN SPRITE

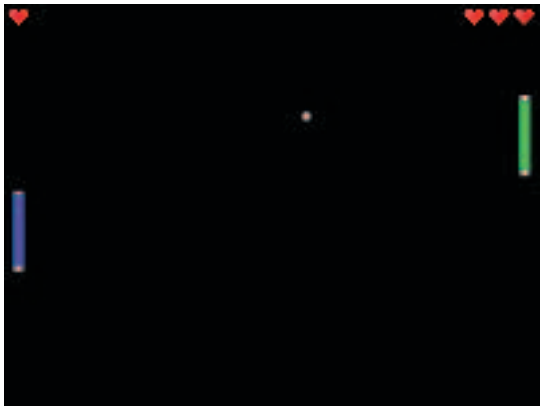
Du har tidigare (i introduktionskapitlet) provat på att visa en bild på skärmen med hjälp av klassen **CDXSurface**. Den används oftast för bakgrundsbilder och andra sorters bilder som inte rör på sig. Saker som rör på sig eller som man interagerar med i ett 2D-spel kallas för sprites, vilka motsvaras av klassen **CDXSprite**. Sprites kan sägas vara spelvärldens innevånare.

Det som huvudsakligen skiljer sprites från ytor (**CDXSurface**) är att de har en position som man kan ändra på, att de kan animeras (se avsnittet *Plattformsspel*) samt att man kan kontrollera om de krockat med varandra. Utöver detta kan man även rotera, förstora och förminska sprites, även om man oftast gör detta i förväg i ett ritprogram i stället.

Drake, bubblor och monster är sprites.



Plattor och boll är sprites.



Figurerna, eldkloten och föremålen är sprites.

A. Din första sprite

Kod: 7.1 – Din första sprite.

Du ska nu testa att skapa en sprite och visa den på skärmen. Tillvägagångssättet känner du igen från introduktionskapitlet, men en del saker är annorlunda så följ dessa steg:

Instruktioner för hur du gör detta finns i del 1, avsnitt 5. Skapa projekt.

1 Nytt projekt

Skapa en ny kopia av **Basecode** och döp om mappen till **Studsboll**, öppna sen **basecode.sln**. Detta steg ska alltid utföras om inte annat anges i instruktionerna till ett exempel, det kommer inte att skrivas med i fortsättningen.

2 Skapa variabel

Tillvägagångssättet när man lägger in någonting nytt i sitt spel är i stora drag detsamma vare sig det handlar om bilder, sprites, kartor eller ljud. Så dessa steg kommer du att stöta på många gånger.

Börja med att skapa en variabel för spriten i **Game.h**. Allt som läggs här blir globalt i spelet, det går att använda från de övriga kodfilerna. Var i **Game.h** du lägger den har ingen större betydelse, bara det är nedanför översta raden **(#include ...)**:

```
//variabel för en sprite
DECLARE CDXSprite* Ball;
```

En variabel som skapas utifrån en klass (**CDXSprite**) kallas mer korrekt för ett objekt. Lite mer om detta kommer du att läsa i avsnittet *Rymdspel*.

3 Ladda bilden

Färgen runtom

...blir genomskinlig.

Fixa en bild till spriten, gärna en boll. Bilden måste vara en bmp-bild som helst inte är allt för stor, förslagsvis ungefär 50x50 pixlar. Spara bilden med namnet **boll.bmp** i projektets mapp (den du skapade i förra steget). Den färg som finns i övre vänstra hörnet kommer att bli genomskinlig.

Inladdning av bilder och liknande sköts i **Init.cpp**. Det som läggs där körs när spelet startar. Leta reda på funktionen **InitGame()** och lägg in följande kod mot slutet av den:

```
//fixa minne till spriten
Ball = new CDXSprite();
//ladda in bilden
Ball->Create(Screen, "boll.bmp", 50, 50, 1);
//ställ in genomskinlig färg
Ball->SetColorKey();
```

Du måste fixa minne till spriten eftersom variabeln är en pekare (med *). Inladdningen är ganska enkel, siffrorna anger bredd, höjd och antal bilder. Om din boll-bild inte är 50x50 pixlar måste du byta ut dessa siffror. **SetColorKey()** ställer in vilken färg som är genomskinlig, i detta fall den färg som finns i bildens övre vänstra hörn. Att få andra färger är ganska knöligt.

4 Visa bilden

Nu återstår uppritning av spriten, all uppritning och allt annat som görs när spelet är igång placerar du i **Active.cpp**. Leta reda på funktionen **Draw()** och lägg in följande kod ovanför **Screen->Flip()**:

```
//rita spriten till skärmen
Ball->Draw(BackBuffer, 0, 0, CDXBLT_TRANS);
```

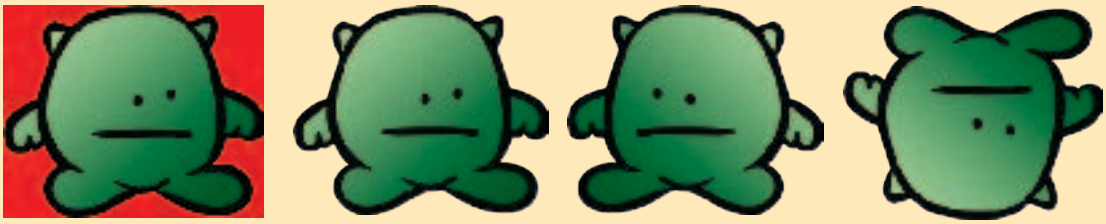
BackBuffer är en yta (en bild) i minnet, allting ritas till den och sen visas den på skärmen (**Screen->Flip()**). Detta kallas för double-buffering och gör att spelet inte flimrar då en hel skärm visas på en gång i stället för att ritas upp lite i taget. Det är en vanlig teknik när man sysslar med grafik, inte bara i spel utan även i vissa användargränssnitt. 0, 0 är inte riktigt var spriten ritas upp, utan snarare var världen befinner sig (om man har en stor scrollande värld). Det sista kryptiska är hur den ritas upp, se informationsrutan nedan.

Olika sätt att rita upp en sprite

I uppritningskoden ovan finns konstanten **CDXBLT_TRANS** med, den anger hur spriten ritas upp. Genom att byta ut värdet kan man rita upp spriten på olika sätt. Prova:

`CDXBLT_BLK`	Som ett block.
`CDXBLT_TRANS`	Med genomskinlig bakgrund.
`CDXBLT_TRANSALPHAFAST`	Delvis genomskinlig.
`CDXBLT_TRANSHFLIP`	Speglad.
`CDXBLT_TRANSVFLIP`	Vänd vertikalt.

Dessutom finns några till som kräver ytterliggare inställningar, till exempel **CDXBLT_TRANSROTOZOOM** som används i avsnittet *Nätverkspel*.



CDXBLT_BLK CDXBLT_TRANS CDXBLT_TRANSHFLIP CDXBLT_TRANSVFLIP

Provkör genom att trycka på  och rätta till eventuella fel (mycket vanligt med fel på `;` eller måsvingar `{ }`). Om ditt spel hänger sig beror det troligen på att det inte kunde ladda in bilden, vanligvis för att den ligger i fel mapp, har fel filnamn eller inte är en bmp-bild. Kontrollera även att måtten stämmer.

Som du säkert genast märkte står spriten väldigt stilla i övre vänstra hörnet. Dags för rörelse!

Fast i hörnet.

B. Den rör på sig

Innan du kan få spriten att röra sig behöver du veta hur du placerar ut den, ställer in dess position. Att få den att röra sig handlar sen bara om att modifiera positionen mellan uppritningarna.

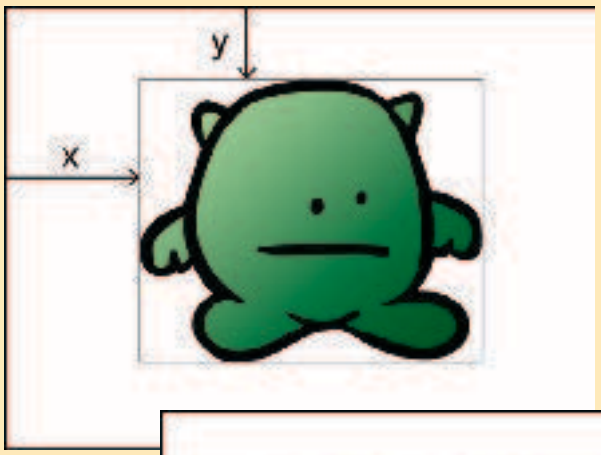
Kod: 7.2 – Den rör på sig.

Allmänt om koordinater och rörelse

Världen i ett 2D-spel (än så länge bara det man ser direkt, skärmen) har ett koordinatsystem med X- och Y-koordinater, ungefär som det du är van vid från matematiken. Allting som ritas ut på skärmen ritas vid någon viss position (X, Y). För att få någonting att röra sig måste man förändra denna position. Om man bara ändrar den en gång så har man flyttat på saken, om man ändrar den löpande får man en rörelse.

Genom att förändra X-koordinaten kan man flytta till höger eller vänster, Y-koordinaten uppåt eller nedåt. Det koordinatsystem som används har origo (0, 0) i det övre vänstra hörnet. Större värden på X betyder längre åt höger, större Y längre ner. Alltså delvis omvänt mot för vad man är van vid från matematiken.

Hur bred och hög skärmen är beror på vilken upplösning du ställt in för spelet. I denna bok används oftast 640 x 480, men om man inte vill låsa koden vid en viss upplösning går det att kontrollera bredden respektive höjden med **Screen->GetWidth()** och **Screen->GetHeight()**.



Övre vänsta hörnet är (0, 0).

Små ändringar av position ger rörelse.

1 Placera spriten

Du kan placera ut spriten genom att använda metoden **SetPos(x, y)** (en metod är en funktion som finns i en klass, i detta fall i klassen **CDXSprite).** Prova så här i **Init.cpp** i funktionen **InitGame()**:

```
Ball->SetPos(50, 50);
```

Var noga med att lägga till koden efter det att du skapat bollen, alltså efter **Ball->Create()**. Annars är inte spriten iordninggjord och det kommer eventuellt inte att fungera som tänkt. Råkar du använda **Ball** innan du fixat minne med **new** får du dessutom minnesfel.

Fast, fast inte i hörnet.

2 Go go go

Om bollen ska kunna åka automatiskt så måste dessa värden ändras hela tiden när spelet är igång. Det gör man enklast genom att kontrollera var bollen befinner sig nu, och sen lägga till eller dra bort något värde. Lägg till följande kod i **Active.cpp**, **UpdateObjects()**:

```
Ball->SetPos(Ball->GetPosX() + 1, Ball->GetPosY() + 1);
```

Ovanstående placerar bollen där den är nu (hämtas med **GetPosX()** och **GetPosY()**) plus en pixel åt höger och en neråt. Resultatet blir en rörelse snett neråt. För att få bollen att åka åt ett annat håll tar man andra värden än + 1. Testa själv!

Gaaaah, it is alive!

Eftersom du inte har någon bakgrundsbild som hela tiden ritar över skärmen blir det spår efter bollen, det löser du enklast genom att tömma skärmen med svart innan du ritar ut bollen. Lägg in följande överst i **Draw()**:

```
//fyll skärmen med svart
BackBuffer->Fill(0);
```

C. Men studsen då?

Kod: 7.3 – Men studsen då.

Din boll tar ingen notis alls om att den passerar kanten på skärmen, utan den fortsätter glatt iväg likt Enterprise till nya outforskade områden. Att få bollen att studsa i stället innefattar två steg:

1. Avgöra om bollen passerar/nuddar kanten på skärmen

Att avgöra om bollen tar i kanten på skärmen gör man genom att kontrollera bollens position mot kantens. Enklast är bollens vänster- och överkant, då det är dessa som anges av dess X- och Y-värden. För att kunna kontrollera under- och högerkanten måste man veta bollens mått (höjd och bredd).

2. I så fall byta riktning på bollen

För att få bollen att byta riktning (studsa) behöver man kunna byta ut värdena som används i **SetPos()** ovan (+1) när spelet är igång. De bör alltså vara variabler, vilka avgör riktning och fart på bollen (dess hastighet). Hastighet heter på engelska velocity och det finns redan färdiga variabler för det i klassen **CDXSprite** (variabler i en klass heter egentligen attribut).

Då bollen först tar i nederkanten visas här hur man fixar den kanten, sen kan du som övning själv ordna övriga kanter. ”Facit” finns på sidan 47. Börja i **InitGame()**, efter att du skapat din boll:

```
//hastighet i x-led respektive y-led
Ball->SetVel(1, 1);
```

I **UpdateObjects()**, ändra den rad du redan har och lägg till övriga:

```
//flytta enligt bollens hastigheter
Ball->SetPos(Ball->GetPosX() + Ball->GetVelX(),
             Ball->GetPosY() + Ball->GetVelY());
//studs mot nederkanten? byt riktning i y-led
if (Ball->GetPosY() + 50 > 480) Ball->SetVelY(-Ball-
>GetVelY());
```

Tar i kanten?

Ovanstående kräver en förklaring, för det första har du **Ball->GetPosY() + 50**. Det är för att man vill kontrollera bollens nederkant, men dess position mäts från överkanten. Bollen är ju 50 pixlar hög. **>480** är för att nederkanten på skärmen i den inställda upplösningen befinner sig på 480 pixlar, större än därför att om man använder **==** måste det vara exakt på pixeln vilket blir problem om bollen flyttar mer än 1 pixel i taget. Det sista inverterar bollens hastighet (hastighet = -hastighet) vilket gör att bollen byter håll. Om bollen till exempel hade hastigheten 3 i y-led (3 pixlar i taget neråt) blir den nu -3 (3 pixlar i taget uppåt).