JavaScript - Examination

Projekt Översikt

Mitt slutprojekt i JavaScript kursen är att återskapa The Legend Of Zelda från NES så nära som möjligt. I detta dokument kommer jag förklara lite mer på djupet om vad jag har gjort i min kod, samt försöka att förklara min tankegång genom projektet.

Det var kul att få använda sina kunskaper man samlat på sig under kursens gång och även kul att få se hur dem fungerar i ett projekt. Det kändes mer som att allt klickade mer när man fick använda kunskaperna i detta projekt, man fick ett bättre perspektiv och en bättre känsla för hur JavaScript fungerar.

Det jag började med att göra var att bryta ner allt i små bitar. Då detta skulle bli ett rätt så sort projekt, så måste man ta en sak i taget, jag bröt ner det på detta vis:

1. Skapa kartan
2. Rita Link och animera honom
3. Kollisionshantering
4. Byta karta och första grottan
5. Plocka upp element
6. Lägga till ljud
7. Lägga till HUD
8. Animera fiender
9. Animera Links attack

Innan vi börjar att gå genom stegen så tänkte jag gå igenom min start kod lite. Det jag börjar med att göra är att definiera en variabel som heter Canvas i den variablen sparar jag min Canvas som jag ska rita ut spelet i DOMen med. Jag hämtar canvasen med document.getElementById.

Sedan definierar jag en variablel som heter ctx och i den så lägger jag min canvas.getContext och det innebär att det jag ritar ut i variabeln canvas är 2d. Variablen ctx representerar mer eller mindre kanvasen i koden.

Efter det så har jag en document body style zoom och det har jag för jag vill återskapa Zelda så nära som möjligt, den originella upplösningen på NES är 250x240px så spritsen är i realisation till den upplösningen så skulle jag sätta en högre upplösning på kanvasen så skulle jag få redigera spritsen så dem stämmer till den nya upplösningen. Så jag tyckte att detta var en enklare lösning.

Efter det så måste jag ladda in min spritesheet in i dokumentet och eftersom det egentligen bara är en bild så skapar jag en variabel som heter worldSprites och lägger in en bild i den variablen, sedan säger jag åt datorn var den ska leta för att hitta den bilden. Sedan definetar jag fps (frames per second).

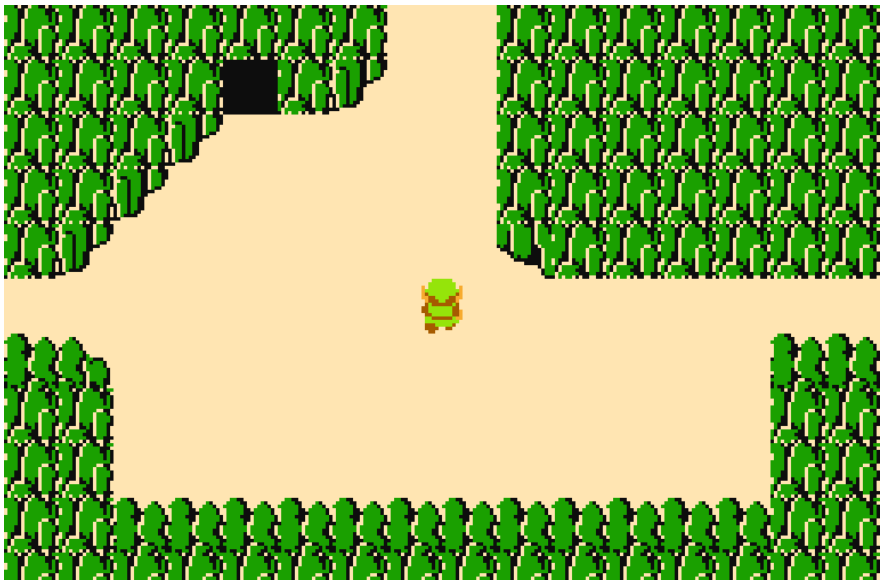
Det sista jag gör innan jag börjar med att skapa kartan är att göra en funktion som skriver ut allt på skärmen.

1. Skapa Kartan

Första steget var att skapa kartan som man startar på. Detta gjorde jag genom att utgå från en så kallad spritesheet och en bild på kartan från spelet.

Första steget till att skapa kartan för start området var att kolla på en bild på kartan och avgöra hur stor kartan är. Tittar man på bilden, så ser man att kartan är indelad i fyrkanter, 16 i bredd och 11 i längd. I projektet så är det 16x15 men de 4 extra raderna är för ”heads up menyn” där man ser sin nivå på hälsan, saker du har i Haden osv. För att rita ut kartan använde jag mig av spritesheets och arrayer, så nästa steg var då att kolla på min spritesheet för att hitta dem sakerna jag ska rita ut på skärmen. Så då kollade jag på spritesheeten som jag ladda ner från google, räkna raderna och kom fram till att det är 17 element i varje rad, så första raden är nr 0 andra nr 18, tredje nr 36 osv.

När jag hade raknat ut hur stor kartan var och vilka nummer på sprites jag skulle använda, så gick jag in i min skript fil och ”ritade” upp kartan genom att använda dem numrera i arrayer



2. Skapa Link och animera honom

Det allra allra första jag började med var att skapa två eventListners som ligger och lyssnar på när en tangent blir nertryckt, en som lyssnar på när den släpps sedan skapar jag även 4 nya variabler, när man trycker uppåt, neråt, vänster och höger, sätter värdet på dessa till false så dem inte blir aktiverade när jag inte trycker på någon tangent. Skapar även en variabel för den sista tangenten som trycktes, värdet på den sätter jag till ”upp” men skulle egentligen ha satt den till något annat värde, det spelar inte så stor roll vad den har för värde, så länge som det är upp, ner, vänster eller höger.

Skapar även en variabel för Link och säger till datorn vart den ska leta efter spritesheetet. En variabel för vart vi är i animationen/hur snabbt animationen ska gå, en variabel för den nuvarande animationen och en för hur snabbt animationen ska röra på sig. Sista variablerna jag gör i detta steg är vart vi vill rita ut Link på skärmen.

När alla mina variabler är skapade så går jag vidare till att skapa funktionen för när man trycker ner en tangent och ger den parametern e, e i JavaScript står för event och det är en del av event handeling functions och där i finns ett bibliotek med massa olika event. I detta fall använder jag e för att kunna använda mig utav keyCode för att kunna säga att det är just piltangenterna vi kommer använda för att styra Link. Sedan kommer en massa if satser som säger om en viss tangent är nertryckt så ska värdet på den ändras från false till true, samt logga vilken tangent som sist trycktes på.

Sedan gör jag en function för när en tangent släpps och den ser ungefär lika dan ut som den funktionen som ändrar värdet i förra steget, det som ändras i denna funktion är att värdet på tangenten blir false igen.

Nästa steg är att rita ut Link. En variabel jag glömde är hur fort Links sprite ska röra sig på skärmen, så den variabeln gör jag i funktionen innan jag går vidare. Man vet inte exakt hur fort Links sprite rör sig i originalet så man får testa sig fram lite vad som ser och känns bra ut, det är många värden som är så. Sedan säger vi att animationCounter, den som styr vart vi är i animationen ska plussa sig själv vare frame. Sedan säger vi om man trycker på den vänstra pil tangenten så ska man röra sig åt vänster i X led, vänster i X ledet är ett negativt värde. Efter det så säger vi om currentAnimation (som styr den nuvarande animationen) är lika med 0 så ska vi rita ut Link som rör sig åt vänster. Parametrarna vi skjuter in där är, vart ifrån ska vi rita Link, vart ifrån på spriteshheten ligger den bilden vi vill skriva ut, i vårt fall 30, 0, 16, 16 (30pixlar in från kanten, 0 pixlar uppifrån och sedan är link en 16x16 sprite), sedan lägger vi till linkX och linkY för att säga till programmet vart på våran skärm vi vill rita ut spriten på, sedan säger vi även att han är 16x16.

Sedan säger vi att vi vill skriva ut bilden (spriten) som ligger 30 pixlar ner. Nästa steg är att göra en if sats som håller koll på animationen så vi faktiskt får en animation. Vi sätter animationCounter >=6, siffran 6 får man fram efter lite experimentering, det är den siffran som känns och ser närmst originalet ut. I denna if sats gör vi så att currentAnimation plussar sig själv och sätter animationCounter till 0 igen, sedan gör vi en till if sats som säger att om currentAnimation är större än 1 så ska vi sätta tillbaka currentAnimation till 0 igen. På så sätt får vi en animation.

Detta gör vi för alla tangenter som trycks ner, upp, vänster och höger. Det vi ändrar är vart datorn ska leta i spritesheetet för den spriten vi ska skriva ut. Sista steget i detta steg är att skriva ut en sprite när vi släppt tangenten, det löser vi med if satser återigen, nu så kommer även den variablen då vi loggar vilken sista tangent som trycktes på in i bilden, vi skapar en if sats och sätter den lika med till ner då ska den skriva ut första spriten man använder när man ska gå neråt, sedan är det samma process för resterande riktningar. Samt så måste vi anropa funktionen så datorn skriver ut Link. Om vi inte anropar funktionen så kommer ej något att hända.

3. Kollisionshantering

För att kunna ha kollisionshantering så måste vi skapa en funktion, i funktionen skickar vi in en parameter för x, en för y och en för vilken karta vi är på. Det man egentligen gör med kollisionshanteringen är att an loopar igenom kartan du är på genom att använda if statemrntd och kollar ifall siffran är tillåten eller ej. Anledningen till att vi kollar om siffran är tillåten eller ej är ju för vi har byggt upp våran karta med siffror.

Det vi gör med if satserna är att vi jämför x och y med elementen på kartan. X och y variablerna är menat inte för inte precis där Link står utan vart han skulle kunna vara, först så kollar vi om det är något som blockerar våran väg, om det inte är det, finns det något annat som skulle kunna vara i vägen.

Sedan för att kollisionshanteringen ska fungera så måste vi lägga in att Link ska röra på sig och det inte är en kollision (&& !kollision) sedan får vi byta ut x och y mot åt vänster så blir det linkX - speed, linkY, map7_7 på sätt kan datorn jämföra om det blir en kollision i x led åt vänster på kartan map7_7. Beroende på vilken karta som karaktären står på så får man ändra kartan bara. Sedan upprepar man bars samma steg för de andra riktningarna, den största skillnaden blir att när man ska upp eller ner så får man ändra så den jämför på y led istället för x.

4. Byta Karta och första grottan

Det första steget var att rita ut den nya kartan över den första grottan. Det gjorde jag på samma sätt som jag ritade ut den första kartan. Sedan definierade jag lite nya variabler, en för varje objekt som kommer finnas, en för varje ny karta, en för den faktiska kartan man är på.

Sedan gick jag vidare till att skapa klassen/ funktionen som ska skapa alla objekt. Normalt sätt när man gör klasser så vill man ha en konstruktör och det är det som funktionen är, en konstruktör som skapar spel objekten, samt brukar man sätta parametrar som definierar variablerna lokalt inom konsrtuktorn, dock eftersom vi kommer ha många olika spelobjekt så väljer jag att göra en generisk funktion och sen definiera allt, istället för att ha miljoner med parametrar som man virrar bort sig i.

Nästa steg är att skapa ingången till grottan. Det vi börjar med är att säga vart på skärmen som ingången till grottan finns, i detta fall 72px i X led och 72px i Y led, sedan hur stor och hög ingången är, dessa värden hitta jag efter lite experimentation, eftersom vi inte vet exakt hur dem gjorde i originalet så får man gå på känsla och hur det ser ut. Sedan vilken karta som ingånen/ portalen tar oss till. Sedan säger vi vart datorn ska skriva ut link på den nya kartan, 120px i X led och 220px i Y led. Sist så säger vi att detta är en portal som ska leda till kartan som har indexvärdet 1 i arrayen, dock kommer den siffran att ändras när fler kartor kommer till, och pushar in allt vi skapat i arrayen för objekten till denna karta (objekt7_7). Sedan för att ”limma” ihop allt så skickar vi både kartan och objekten till en klass vi gjorde tidigare som parametrar, detta kommer vi göra för varje karta och tillhörande objekt.

Sedan så måste vi ju skapa en utgång också, det gör vi på samma sätt som vi gjorde ingången, men vi får byta lite siffror, X blir nu 112px , Y blir nu 240px, höjden och brädden blir nu 16x16px och så ska den leda tillbaka till start kartan som har indexvärdet 0 i arrayen. Sedan pushar in allt vi skapat i arrayen för objekten till denna karta (spelObjektC1), ”limmar ihop” allt och skickar det till klassen.

Nu när vi har skapat alla objekten vill vi också kunna kollidera med dem, så för att kunna göra det så måste vi skapa en funktion som hanterar kollisionen med våra objekt i spelet. Denna funktion kommer växa under skapandets gång, för här kommer vi lägga in och kolla ifall vi stöter i någon fiende, vart svärdet slår i etc, men just nu så behöver vi bara parametrar för, x, y, objekten och om det är link. Så det vi börjar med är att göra en if statement som kollar om det ät link eller inte, sedan gör vi en for loop som loopar igenom alla objekten. Det vi gör sedan är att kolla om det sker en kollision och vad vi kolliderar med isåfall.

Sedan för vi ska kunna gå inom den nya kartan så måste vi säga att nummer 28 är ”godkänd” detta gör vi i funktionen som kollar om något är vägen för spriten, vi lägger bara till && map[j][] != 28 i if statementet. Samt så får vi byta ut map7_7 till våran dynamiska karta spelKarta

7. Lägga till hud

När det kom till att lägga till rubinernas antal, så ville jag ha det så att när man har mindre än 100 rubiner ska x:et innan synas men inte om man har mer än 100. Så jag började med att göra en if sats som tog hand om då man har mindre än 100st. För att få entalets siffra så gjorde jag en (let variabel för denna siffra kommer skrivas över) variabel som tar antalet rubiner du har och tar det som blir kvar när man delar den siffran med 10 och sparar det i variabeln firstNum. Sedan så måste jag berätta för datorn, vart den ska hämta ifrån på spritesheeten och vart på skärmen den ska skriva ut spriten. Så i från variablen hud, ska den börja leta på 528px (vilket är nummer 0), sedan beroende på hur många rubiner du har så ändras den siffran och på så sätt skriver ut olika siffror på entalets plats. Sedan för att skriva ut tiotalet så gör man på liknande sätt, ändrar bara variabelnamnet till secondNum så vi inte skriver över fel variabel, lägger till Math.floor (så vi får en hel siffra) på rupeeAmount och delar den med tio och sist så säger vi att den ska skrivas ut på 8 istället för 16.

Sedan för att lägga till nummer som är över 100 så var det bara att göra en else stats och utgå ifrån den koden som fanns i if satsen. Det som blir annorlunda är att man lägger till ett tredje nummer och ändrar uträkningen för det andra numret. Annars är koderna rätt lika varandra.

Sedan så måste vi skriva ut antalet nycklar och bomber vi har, det gör vi genom att utgå ifrån den koden vi skrev när vi skrev ut antalet rubiner. Det som vi ändrar är vart på skärmen det ska skrivas ut.

Nästa steg i att fylla ut huden är att skriva ut svärdet, det vi gör då är att vi använder den koden som vi använder för att täcka över de ”tomma pixlarna” och täcker över där det vi har i handen syns, sedan gör vi en if sats som säger om svärdet är utrustat så ska vi skriva ut svärdet

8. Animera Fiender

Innan vi börjar att animera fienderna måste vi börja med att lägga till nya variabler i våra spelobjekt, vi behöver en som säger är det en fiende och vilken fiendetyp det är. Jag lägger även in andra variabler som vi kommer använda senare i projektet som har att göra med hur finderna rör sig. Den första är nästa X position, nästa är en för nästa Y position, en som säger om fienden attackerar eller inte, vi vill veta fiendens hälsa, vi vill veta vilket håll som fienden är riktad mot, när vi väll lägger in rörelse på fienden så kommer vi vilja veta närmsta vägen till den slumpade koordinaten som den valt, så då skapar vi en array som vi lägger in allt i sen, eftersom vi kommer animera fienden så behöver vi en räknare och Frame, vi behöver veta om när link attackerar fienden, om den ska flytta på sig eller inte och då behöver vi givetvis veta hur mycket i X och Y den ska flytta sig.

Innan vi börjar och animera fienden, så måste vi placera ut en fiende. Det gör vi genom att hitta koden till den karta som vi vill placera ut fienden i, i detta fall så placerar jag ut fienden i starter kartan och då utgår jag från koden som skapar ingången till grottan men gör lite modifikationer så den ska skapa ett fiende objekt och lägga in det i arrayen.

Sedan måste vi skriva ut fienden också, då får vi gå in i funktionen som skriver ut spel objekt och gör ett if statment som säger om det är fiende så ska detta ske. Det vi börjar med där är att iterera räknaren, sedan säger vi om räknaren är större eller lika med 10 så ska vi skriva ut en ny Frame och sätta räknaren till 0 igen, så vi kan animera fienden. Här skulle man kunna använda sig utav switch statements för det kommer bli mycket fienden, men nu när vi bara har en så gör vi så här. Olika fiender kommer ha olika antal frames, just denna har två stycken, Frame 0 och frame 1, det vi vill göra nu är att när såfort som framen går över 1 så, ska den sluta skriva ut och vi nollställer framen igen.

Nu vill vi skriva ut fienden i alla riktningar, så vi kikar på fiendens sprites och ser att den har 8 sprites och två i varje riktning. I koden så gör vi ett if statement som hanterar varje direktion och i den så skriver vi två if-statements som skriver ut fienden vid frame 0 och frame 1. Detta gör vi för alla riktningar

9. Animera Link Attack

Innan vi börjar animera attacken, så måste vi återigen definiera nya variabler, en som säger om Link har sitt svärd eller inte, en som säger om han attackerar och en som säger om han kan attackera igen. Sedan vill vi också koppla allt detta till en knapp, jag valde att koppla attacken till spacebar.

Sedan går vi till funktionen som hanterar links animationer och säger om link attackerar och har svärdet. Om den nuvarande animation är 0 så ska vi hämta den sista knappen vi tryckte på, för det kommer säga oss åt vilket håll Links attack bör komma från och sedan skriva ut den spriten för varje riktning. Sedan så säger vi om den nuvarande animationen är 1 så ska vi skriva ut spriten med svärdet också. Efter som spriten med svärdet lägger på nya dimensioner på spriten så måste vi ändra lite mer i koden för att ha kvar link på samma ställe som han står på när han attacker, annars kommer han flytta på sig. Så på upp attacken säger vi linkY-14, på vänster får vi kompensera både X och Y led, så link X - 10 och linkY - 8 och åt höger får vi kompensera på Y led med - 8.

Det vi vill göra sedan är att vi säger om animations räknaren är större eller lika med 6 sp ska den nu varande animationen itereras, animations räknaren ska tillbaka till noll och om den nuvarande animationen är 1 så ska den nuvarande aminationen tillbaka till noll, att link attackerar blir false och att link kan attackera igen blir true.