Arbetsrapport

2021-05-30

1ME322 - Webbteknik 2 Felix Sopelsa fn222hn@lnu.se

Det första steget som togs för att lösa denna uppgift var att sätta mig in i hur programmet är tänkt att fungera. Detta gjorde jag genom att noggrant läsa igenom uppgiftsbeskrivningen och dessutom skriva ner de punkter som fanns där med kommentarer på papper för att sedan kunna checka av dessa när de är gjorda. Sedan satte jag mig in i både HTML-koden och CSS-koden för att få en övergripande bild av hur de funktioner som sedan ska skrivas kan komma att fungera. Nu kunde jag börja försöka översätta de punkter jag tidigare skrivit till javascript och detta gjorde jag genom att skriva en lista med preliminära funktioner samt kortfattat vad som ska hända inom varje funktion. Eftersom att jag så noggrant gått igenom vad som ska hända i programmet innan jag började skriva kod så hade jag kommit fram till att det borde gå bra att skriva programmet i samma ordning som uppgiftsbeskrivningen var skriven.

Jag började alltså med att skriva en init-funktion och där lägga in de referenser till HTML-koden som jag från början visste skulle behöva finnas. Jag visste också att knappen "nytt spel" skulle aktiveras och att knappen "nya brickor" inaktiveras vid init, samt att dessa behöver händelsehanterare, så detta kunde jag programmera redan nu.

Sedan skrev jag nästa funktion som jag valt att kalla newGame. Denna funktion händer när man trycker på knappen "nytt spel" och då avaktiveras den knappen medan knappen "nya brickor" aktiveras. Efter att ha gjort detta så lämnade jag newGame så länge.

Istället började jag skriva på nästa funktion för att kunna få igång och kunna börja jobba med själva spelet. Jag började alltså skriva funktionen newBricks vars uppgift är att slumpa fram 4 brickor och lägga dessa i HTML-elementet "newBricks". Detta händer genom att använda tekniken Math.floor(tempList.length*Math.random()) som alltså slumpar fram 4 bilder från arrayen templist, (en kopia av den globala arrayen allBrickNames som innehåller alla nummer mellan 1-40). När fyra nummer slumpats fram så tas de bort ifrån tempList så att samma bilder inte ska kunna slumpas fram igen. Kopieringen av allBrickNames görs varje gång ett nytt spel startas. I newBricks-funktionen får de 4 nya brickorna också händelsehanterare och blir dragbara, samt får klassen brick istället för empty. När 4 brickor slumpats fram inaktiveras "Nya brickor"-knappen tills de fyra brickorna dragits till spelbrädet.

_____När detta gjorts i newBricks så var det dags att skriva funktionerna dragstartBrick och dragendBrick. Dessa ser till så att det går att att släppa brickorna på tomma rutor i brädet, och att det inte går att dra tomma rutor från newBricks. I dragstartBrick läggs dessutom både själva bilden och bildens id in i dataTransfer så att de följer med till där de sedan släpps på brädet.

Sedan började jag skriva funktionen som aktiveras när man drar en bild till brädet, nämligen brickOverBoard. Här bestäms vad som händer när man släpper en bild på brädet,- först och främst så blir rutan som man dragit bilden ifrån tom samt får klassen empty istället för brick. Genom dataTransfer.getData fastnar sedan både bilden och bildens id i den nya rutan som man nu släppt den i. Dessutom får denna ruta klassen brick istället för empty. I funktionen brickOverBoard bestäms också att de rutor man drar bilden över, före man släppt den, får en grå färg tills dess att man drar ut bilden från rutan.

Nu var det dags att skriva en funktion som undersöker huruvida alla rutor för nya brickor är tomma samt en funktion som gör samma sak fast med brädet. Dessa kallar jag controlForNewBricks och checkIfBoardIsFull. I dessa funktioner kontrollerade jag först detta genom if- och else-satser som undersökte hur många rutor som hade klassen empty. Jag hade i det här skedet bara tagit fram en referens till alla rutor i brädet och en till alla rutor för nya brickor och detta fungerade till en början som jag ville. Men vid ett senare skede i arbetet med programmet så stötte jag på problem och efter att ha bett om hjälp från lärare fick jag tipset att göra en ny referens för endast de rutor i brädet som har klassen empty och

detsamma för nya-brickor-rutorna. Nu kunde jag alltså istället bara kontrollera längden på newBrickElemEmpty och boardElemEmpty, om dessa 4 respektive 16 så är de tomma. Så om newBrickElemEmpty är 4 aktiveras "nya brickor"-knappen igen i funktionen controlForNewBricks, och om boardElemEmpty är 0 så avaktiveras samma knapp i funktionen checkIfBoardIsFull, och funktionen som ska räkna ut poäng anropas.

Nu var det alltså dags att räkna ut poäng och detta gör jag i funktionen countPoints. Denna funktion är nog den som har varit svårast att få till på ett smidigt sätt och den har skrivits om och optimerats ett antal gånger. Till en början skrev jag ett väldigt långt stycke kod som kontrollerade stigande ordning för en rad eller kolumn i taget. Detta fick jag inte till så att det fungerade bra, och jag fick be om hjälp av lärare eftersom jag inte lyckades omvandla bildernas id till nummer som går att jämföra i if-satser. Just det problemet med omvandling till nummer löste jag dock under tiden jag väntade på svar, men med de tips som sedan kom som svar kunde jag optimera koden ytterligare. Poängberäkningen sker på så sätt att om det stämmer att rutorna i en rad eller kolumn har en stigande ordning så ökar variabeln pointsThisGame med 1. När detta stämmer så skrivs dessutom en bock ut jämte raden eller under kolumnen i fråga, och stämmer det inte så skrivs iställer ett kryss ut.

När alla rader och kolumner kontrollerats så anropas funktionen showPoints. Här skrivs omgångens poäng ut tillsammans med passande text under spelbrädet och knappen för nytt spel aktiveras igen så att man kan spela igen. Här skickas också omgångens poäng till arrayen totalPoints för att senare kunna adderas ihop med nya omgångars poäng till en totalpoäng. Denna funktionen har jag återkommit till vid senare tillfälle för att spara totalpoängen i localStorage. Till en början försökte jag nämligen göra det inom showPoints, men efter ha gått tillbaka och tittat på laborationen som handlar om localStorage kom jag fram till att detta nog borde göras i en annan funktion som dessutom ligger i ett annat javascript-dokument.

Nu tog jag alltså den javascript-fil som fanns med i uppgiftens arbetsmaterial -cookies.js och döpte om den till localStorage.js. De funktioner som fanns i filen fick nya namn och jag började testa olika metoder för att spara totalpoängen i localStorage. Just den biten är inte speciellt avancerad utan problem uppstod snarare när poängen i localStorage ska läggas ihop med nya poäng från en ny runda efter att sidan uppdaterats. Jag hade hittat en teknik på w3schools.com för att räkna ut summan av flera tal i en array, nämligen metoden .reduce. Denna kokar ner arrayens alla tal till ett genom att först anropa en annan funktion - getSum som använder total + Math.round(num) för att addera ihop dessa och skicka tillbaka summan. Denna metoden använde jag alltså redan för att räkna ihop poäng från flera rundor, så tanken var att göra på samma sätt för att addera poäng från flera sessioner. Därför gjorde jag en ny array där jag efter varje runda la in den nya totalpoängen, för att sedan koka ner summan av den till localStorage. Men detta visade sig va lättare sagt än gjort och jag fick i flera timmar sitta och testa olika varianter av detta och leta syftningsfel etc. Men efter att ha sovit på saken kom jag fram till att jag inte behöver en ny array. Istället använder jag mig av totalPoints-arrayen som jag redan har. Nu sparas summan av totalPoints i localStorage efter varje spel så att arrayen kan tömmas. Sedan läggs den totalpoäng som finns localStorage in igen i arrayen varje gång ett nytt spel startas. På så sätt blir den tidigare totalpoängen ett av de tal som sedan adderas ihop till en ny totalpoäng varje gång ett spel avslutas.

Efter detta så var det bara en programfunktion kvar att implementera. Antalet startade spel ska räknas upp och skrivas ut. Detta gjordes hyfsat enkelt genom att lägga på +1 varje gång man trycker på "nytt spel" och spara i localStorage.

Efter ett zoom-möte med muntlig genomgång av programmet med lärare, fick jag tips om hur countPoints kunde optimeras ännu mer genom for-loopar där c = document.getElementsByClassName("c"+[i]). Nu gås alltså alla kolumner (c1 till c4) igenom i samma loop, och detsamma görs för raderna. Som det nu är programmerat så består

alltså hela funktionen countPoints av två stycken for-loopar istället för de åtta loopar som jag från början skrivit för att kontrollera stigande ordning och räkna poäng.