Webbsida för Sudokuspel

Av Assar Markström

Sammanfattning

Programmet låter användaren lösa Sudokun av önskad svårighetsgrad. På "startsidan" kan användaren välja svårighetsgrad för sudokut. Utöver detta kan användaren läsa lite kortfattat om spelets regler, spelets historia, speltips eller lite kuriosa om sidans skapare. När användaren valt svårighetsgrad och tryckt på startknappen omdirigeras användaren till en ny webbsida där ett sudoku-bräde presenteras. Sudoku-brädets generering är direkt kopplat till vald svårighetsgrad. På själva" spelsidan" finns det ytterligare två knappar. Den ena tar användaren direkt tillbaka till "startsidan och den andra knappen, "reload", genererar ett nytt sudoku-bräde av samma svårighetsgrad. Användaren löser sudokut genom att trycka på en tom ruta och skriva en rätt siffra. Om fel siffra anges varnas användaren och rutan förblir tom.

Framtagande

Ursprungligt koncept och planering:

Grunden för webbplatsen “Sudoku Game” kom från en idé att bygga vidare på ett tidigare arbete och skapa en mer omfattande Sudoku-upplevelse. I en tidigare kurs: Programmeringsparadigm fick jag i uppgift att utveckla en “Sudoku-solver”. Jag kände att det var ett perfekt tillfälle att lyfta mitt gamla projekt genom att integrera ytterligare funktioner och på så sätt göra det till en färdig produkt. Idén tog verkligen form när jag insåg att jag lätt kunde utöka programmet med en Sudoku-pusselgenerator. Detta och en dedikerad startsida förvandlade mitt ganska begränsade projekt till en färdig spelplattform för Sudoku.

Med denna vision i åtanke påbörjade jag planeringsfasen av projektet. Det ursprungliga konceptet gick ut på att ge användaren en enkel och lätthanterlig sudoku lösnings-upplevelse. En central vision var möjligheten att generera nya pussel av varierande svårighetsgrad. I mitt föregående projekt var olika bräden “hårdkodat”. Tillägget av en startsida syftade till att ytterligare etablera känslan av en färdig produkt samt ge utrymme för framtida tillägg.

Under planeringsfasen var jag tydlig med att identifiera de viktigaste målen och de nödvändiga komponenterna för att kunna förverkliga projektet. Eftersom mina kunskaper inom webbutveckling och Javascript är något begränsade var detta oerhört viktigt för att inte bli överväldigad av alla funktioner och tillägg som skulle kommas att implementeras. Att kartlägga användares resa från startsidan till att ha ett löst Sudoku-pussel och sedan följa denna resa i utvecklingsprocessen gav mig en tydlig start- och slutpunkt. Sammanfattningsvis lade den inledande koncept- och planeringsfasen grunden för utvecklingen av webbplattformen och skapade förutsättningar för en omfattande och engagerande Sudoku-upplevelse.

Kodbas och implementering:

Med hjälp av kunskaperna och koden från kursen Programmeringsparadigm började jag med att verifiera min implementering av den grundläggande algoritmen för Sudokulösning i JavaScript. Grunden till min Sudoku-validering kan ses på bilden nedan:

```
// Compute all 27 blocks of a sudoku. Returns all 9 rows, 9 cols and 9
// 3x3 blocks
function blocks(sud) {

    var colBlock = emptySudoku();
    var boxBlock = emptySudoku();

    // rows
    var rowBlock = sud;

    // cols
    for (var i = 0; i < sud.length; i++) {
        for (var j = 0; j < sud.length; j++) {
            colBlock[i][j] = sud[j][i];
        }
    }

    /* We go through all 9, 3x3 "boxes", and put all the elements from the box,
    in an array of 9 elements so that we get a 9x9 array.
    Indices for sud is "block-wise" and indices for boxBlock is "array-wise".
    */

    for (var lc = 0; lc < sud.length; lc += 3) { // lc = large col := 3 cols
        for (var lr = 0; lr < sud.length; lr += 3) { // lr = large row := 3 rows
            for (var r = 0; r < 3; r++) { // row
                for (var c = 0; c < 3; c++) { // col
                    boxBlock[(lr / 3 + lc)][(r * 3 + c)] = sud[(r + lr)][(c + lc)];
                }
            }
        }
    }

    return rowBlock.concat(colBlock).concat(boxBlock)
}
```

Algoritmen börjar, med hjälp av funktionen blocks (bilden ovan), att lista alla rader, alla kolumner och alla 3x3 “block” i en lång lista. Därefter verifieras dessa 27 “block” genom att kolla så att inget block innehåller en siffra (1-9) mer än en gång. På så vis verifieras det att alla rader kolumner och 3x3 “block” är godkända varje gång en ny siffra ska läggas till. Den tidigare koden inkluderade även en mängd testfall vilket underlättade valideringen av algoritmens korrekthet och tillförlitlighet.

Utökning med Sudoku-generator:

Efter att säkerställt implementationen av “lösaren” utökade jag projektet till att omfatta en Sudoku-pusselgenerator. Detta innebar att utveckla en backtracking-algoritm för att generera giltiga Sudoku-pussel.

```
// Generating Sudoku board using backtracking
function generate(sud) {
  for (var row = 0; row < 9; row++) {
    for (var col = 0; col < 9; col++) {
      if (sud[row][col] === 0) {
        const array = [1, 2, 3, 4, 5, 6, 7, 8, 9];
        const nums = array.sort(() => Math.random() - 0.5);
        for (var num of nums) {
          sud[row][col] = num;
          if (isValid(sud)) {
            if (generate(sud)) {
              return true;
            }
          } else {
            sud[row][col] = 0;
          }
        }
        return false;
      }
    }
  }
  return true;
}
```

Implementationen är ganska okomplicerad:

- Välj ut ett nummer från en lista med nummer 1-9 (ej ordnade)
- Lägg till siffran och validera det nya brädet.
- Om det nya sudoku brädet inte är giltigt, gå tillbaka ett steg och ta en ny siffra.
- Fortsätt så tills varje plats på brädet är ifyllt.

Nästa steg av processen var att kunna ha olika svårighetsgrader vid brädets generering, såsom lätt, medium, svår och expert. Detta löstes genom att återigen iterera över hela brädet och för en given sannolikhet (remove Probability), ta bort siffran. Sannolikheten att ta bort siffran är direkt kopplad till svårighetsgraden, vilket innebär att en svårare svårighetsgrad medför en högre “remove probability" och tvärtom.

Integration med webbplattform:

Med de grundläggande funktionerna och logiken på plats fortsatte jag med att integrera Sudoku-lösaren och Sudoku- generatorn i en webbplattform. En primitiv “startsida” designades med HTML och CSS. Målet med designen för användargränssnittet var att den skulle vara visuellt tilltalande och skapa intuitiv upplevelse för användarna. Hanteringen av användar-interaktioner sett till sudoku lösnings-processen var till stor del klar och mycket fokus lades istället på användar-interaktioner för “startsidan”.

Testning och vidareutveckling:

Omfattande tester genomfördes för att identifiera och lösa de buggar och problem som fanns i applikationen. Ett problem som tacklades på olika sätt under utvecklingsfasen var hur den valda svårighetsgraden skulle hanteras. Jag försökte få lokal lagring att fungera med redirect men insåg att det var enklare att bara skicka informationen direkt i url:en, vilket kanske inte är optimalt ur säkerhetssynpunkt men fungerade. Feedback från en kompis som också går denna kurs användes för att förbättra eventuella brister i användargränssnittet. Den tidigare kodbasen ändrades frekvent för att förbättras och anpassas till det nya objektet.

Form

Startsida:

För DOMContentLoaded definieras två primära funktioner: hantering av klick-händelsen för start-knappen och inställning av event-lyssnare för knapparna för svårighetsgrad. När du klickar på knappen "Start" triggar en event-lyssnare den tillhörande callback-funktionen. I denna funktion kontrolleras först om en svårighetsgrad har valts genom att söka efter ett element med klassen "selected" bland ".difficulty-btn"-elementen. Om en svårighetsgrad har valts konstruerar skriptet en URL som innehåller den valda svårighetsgraden som en frågeparameter och omdirigerar användaren till sidan "solveSudoku.html" med den valda svårighetsgraden kodad i URL:en. Om ingen svårighets-knapp är intryckt när "Start"-knappen klickas, används en blinkande effekt på alla “svårighets”-knappar för att dra till sig användarens uppmärksamhet. Händelselyssnare konfigureras för varje “svårighets”-knapp för att spåra användarens klick. När användaren sedan klickar på en “svårighets”-knapp tar skriptet bort klassen "selected" från alla knappar och lägger endast till den på den inklickade knappen, vilket visuellt anger den valda svårighetsnivån.

Pseudokod för “startsida”:

On DOMContentLoaded:
    Set up event listeners for the "Start" button and “difficulty selection” buttons.

Start Button Click Event Handler:
    When the "Start" button is clicked:
        If a difficulty level is selected:
            Construct URL with selected difficulty as query parameter.
            Redirect user to "solveSudoku.html" page with encoded difficulty level in the URL.
        Else:
            Apply flashing effect to unselected difficulty buttons.

Difficulty Button Event Listeners:
    For each difficulty button:
        Add click event listener:
            Remove "selected" class from all buttons.
            Add "selected" class to the clicked button.

Sudoku-sida:

Sudoku-sidan är i själva verket hela Sudoku-spelet.

JS-Skriptet börjar med att koppla en event-lyssnare till DOMContentLoaded-händelsen, vilket säkerställer att koden körs när HTML-innehållet är helt laddat. Här hämtas den valda svårighetsgraden från URL-fråge parametrarna med hjälp av URL Search Params.

Ett tomt Sudoku-bräde initialiseras med funktionen emptySudoku(). Funktionen generate() genererar ett komplett Sudoku-pussel med hjälp av en backtracking-algoritm och funktionen blocks() vilket förklaras i avsnittet Framtagande, mer specifikt: Utökning med Sudoku-generator. Siffror tas sedan bort från det genererade Sudoku-brätet baserat på den valda svårighetsgraden med hjälp av funktionen removeNumbers(). Funktionen init() konfigurerar den första visningen av Sudoku-brädet, fyller celler som inte är noll med värden och konfigurerar klick händelser för tomma celler.

En stor del av användarinteraktion styrs av funktionen klick() som aktiveras när en användare klickar på en tom cell i Sudoku-brädet. Funktionen klick() uppmanar användaren att ange ett värde för den valda cellen, validerar inmatningen och

uppdaterar cellen i enlighet med detta. Ogiltiga inmatningar leder till varningsmeddelanden för att vägleda användaren och säkerställa att Sudoku-reglerna följs. Om Sudoku-pusslet löses efter en användarinmatning visas ett avslutande meddelande och alla celler fylls med samma bakgrundsfärg för att ytterligare visa att sudokut är löst.

För sudoku-valideringen används funktionen isValidBlock() som kontrollerar giltigheten för ett block (rad, kolumn eller 3x3 “subgrid”) i Sudoku-brädet och ser till att det inte finns några duplicerade nummer. Funktionen isValidBlock() har en “förälder”-funktion, funktionen isValid() som kontrollerar giltigheten för hela Sudoku-brädet genom att validera alla block. Funktionen isSolved() avgör om Sudoku-pusslet är helt löst genom att säkerställa att det inte finns några tomma celler och validera hela rutnätet.

Funktionen getSudoku() hämtar den aktuella statusen för Sudoku-brädet från HTML-elementen, och uppdaterar brädet.

Pseudokod för “sudoku-sida”:

On DOMContentLoaded:
    Retrieve selected difficulty from URL parameters.
    Set up event listeners for "Refresh" and "Home" buttons.
    Initialize an empty Sudoku grid.
    Generate a complete Sudoku puzzle.
    Remove numbers from the generated puzzle based on difficulty level.
    Set up the initial display of the Sudoku grid.

Function klick():
    Prompt user to input a value for the clicked cell.
    Validate user input:
        - Check for the cancel button or empty input.
        - Convert input to number and check for validity.
        - Update cell if input is valid.
        - Display alert messages for invalid input.
        - Check if the Sudoku puzzle is solved after user input.

Function isValidBlock():
    Check if a block (row, column, or 3x3 subgrid) in the Sudoku grid is valid.

Function isValid():
    Check the validity of the entire Sudoku grid by validating all blocks.

Function isSolved():

Determine if the Sudoku puzzle is completely solved by ensuring no empty cells and validating the entire grid.

Function getSudoku():

Retrieve the current state of the Sudoku grid from the HTML elements.

Funktion

Det färdiga programmet består av två stycken huvudsidor: startsidan och Sudoku-sidan.

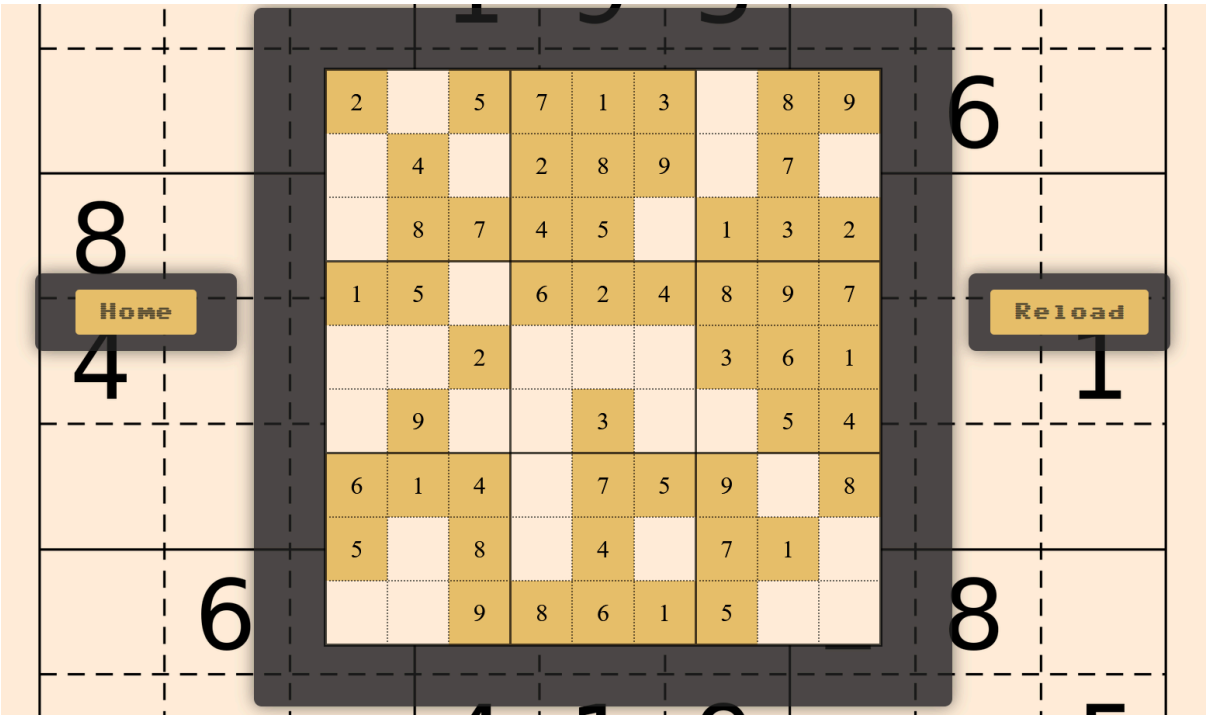
Startsida:

I bilden ovan visas huvudsidan för programmet. Den har en container som är uppdelad i två sektioner: en vänster container och en höger container. Den vänstra “containern” innehåller knappar för att välja svårighetsgrad på Sudoku-spelet och lite information om grundaren och spelet. Användaren kan även välja att kontakta sidans grundare om de skulle vilja det. I den högra “containern” finns sektioner med information om spelregler, historia och tips.

Funktionalitet:

Användaren kan välja svårighetsgrad (Easy, Medium, Hard eller Expert) genom att klicka på motsvarande knapp. Genom att klicka på Start-knappen omdirigeras användaren till Sudoku-sidan med den valda svårighetsgraden. Om användaren försöker starta spelet utan att välja svårighetsgrad blinkar knapparna för svårighetsgrad för att uppmana användaren att göra ett val.

Sudoku-sida:

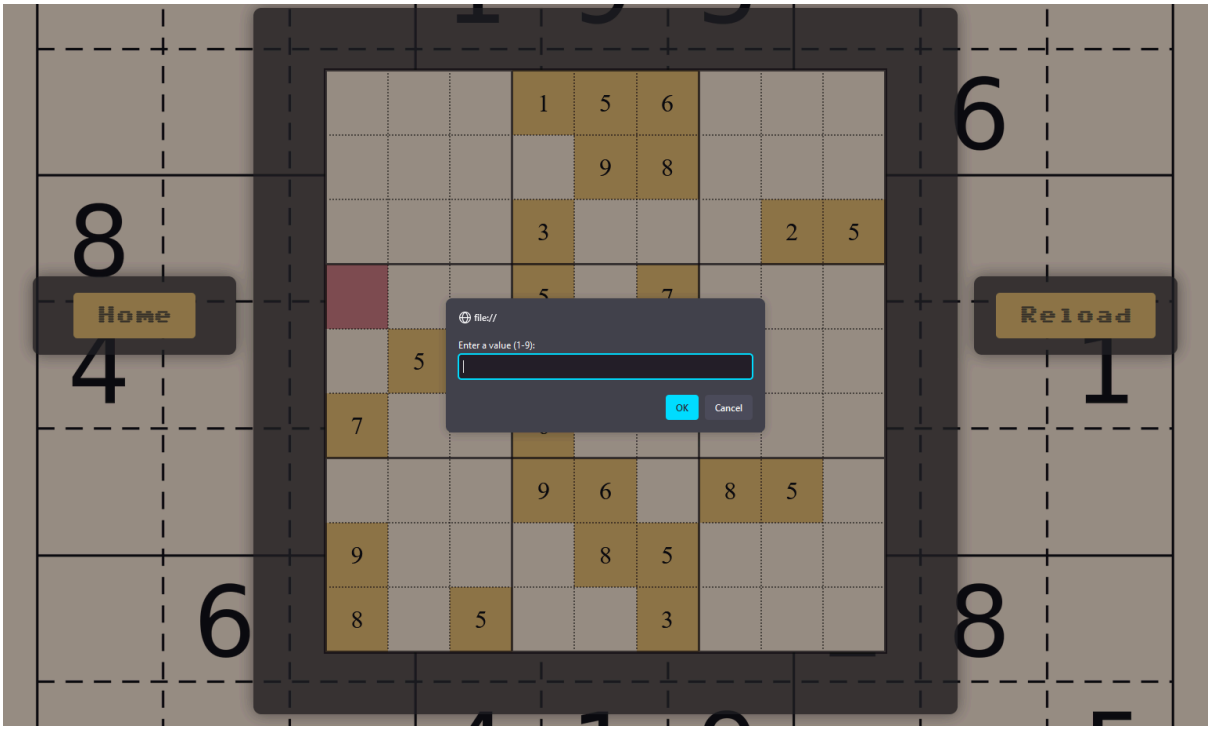


I bilden ovan visas Sudoku-sidan med “Medium”-svårighetsgrad vald. Sidan har i mitten en rutnät-baserad layout som representerar en Sudoku-tavla. Den består av ett 9x9-bord där användare kan mata in siffror för att lösa Sudoku-pusslet. Det finns också två knappar: en för att uppdatera Sudoku-pusslet och en annan för att återgå till huvudsidan.

Funktionalitet:

Svårighetsgraden som väljs på huvudsidan bestämmer den initiala konfigurationen av Sudoku-pusslet. Användaren kan mata in siffror i tomma celler i Sudoku-brädet genom att klicka på dem och ange ett tal mellan 1 och 9 i en prompt. Programmet validerar användarens inmatningar enligt Sudoku-reglerna och ger feedback vid ogiltiga inmatningar. Om användaren lyckas lösa Sudoku-pusslet visas ett gratulations meddelande och hela brädet fylls med samma bakgrundsfärg. Användaren kan uppdatera Sudoku-pusslet eller återgå till huvudsidan med hjälp av

respektive knapp. Om uppdateringsknappen trycks genereras ett nytt sudoku-bräde av samma svårighetsgrad.



I bilden ovan visas Sudoku-sidan med “Expert”-svårighetsgrad vald. I bilden kan man även se hur användar-interaktionen hanteras ifall man väljer att klicka en tom ruta på spelbrädet.

Det ska tilläggas att den nuvarande implementationen, genom att förhindra inmatning av ogiltiga siffror och se till att sudokut behåller ett giltigt tillstånd under hela lösningsprocessen, inte förhindrar brute-force-lösning av pusslet. Eftersom felaktiga inmatningar avvisas säkerställer den endast att giltiga inmatningar görs. Den tvingar inte fram en specifik lösningsstrategi eller hindrar användaren från att prova flera nummer tills ett giltigt hittas. Därför kan en beslutsam användare fortfarande potentiellt brute-forcea sig igenom pusslet genom att prova olika siffror tills rätt siffra hittas för varje cell.

Det fina med Sudoku ligger i det logiska och strategiska tänkandet som krävs för att komma fram till en unik lösning. Om man väljer att brute-force-lösa sudokut går man miste om den tillfredsställelse och mentala träning som kommer av att lösa pusslet genom logik och resonemang.