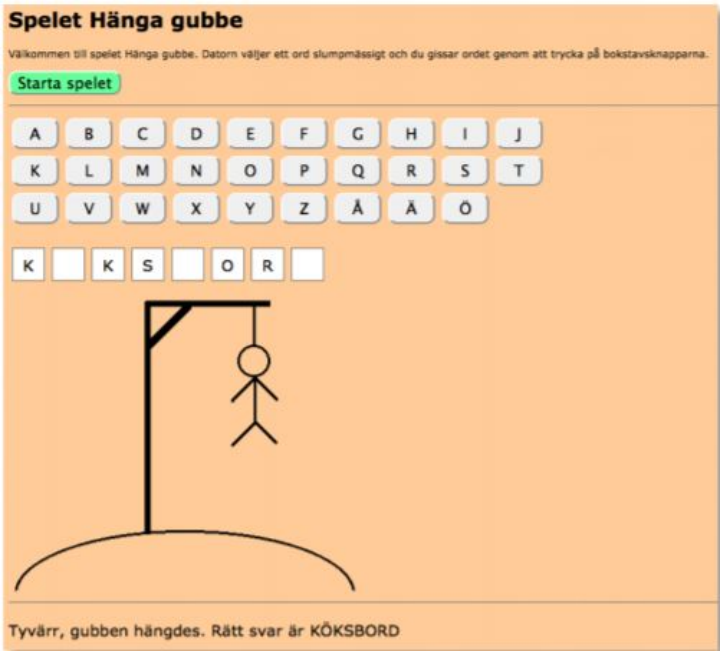
Uppgift 9 - Objekt i Javascript

Planering och struktur för programmet

I uppgiften ska du skapa ett program för spelet Hänga gubbe. Det går ut på att ett ord väljs slumpmässigt och du kan sedan gissa på bokstäver. Varje gång du gissar på en bokstav som inte ingår i ordet, utökas en bild, så att det till slut blir en hängd gubbe. Du kan totalt gissa fel sex gånger, innan gubben hängs. Men lyckas du få alla bokstäver rätt innan dess, klarar sig gubben.
Programmet består av ett antal steg och det blir då också några funktioner för dessa. Innan man börjar skriva koden, är det bra att planera vad som ska göras och skapa en struktur för programmet. Det beskrivs här, innan vi kommer in på övningarna med kodskrivandet.

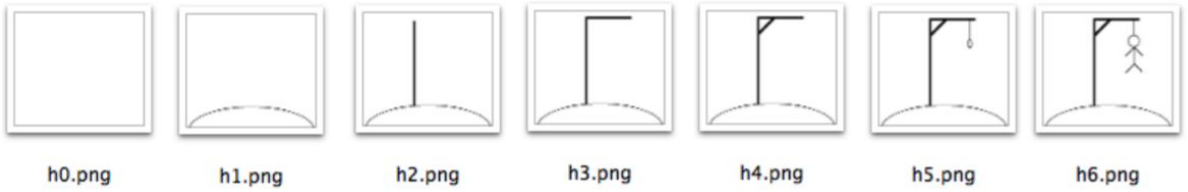
Studera gränssnittet

- Öppna filen *index.html* både i webbläsaren och i editorn.
- Det finns ett antal delar i gränssnittet:
    - Startknapp, för att starta spelet.
    - Div-element "gameBoard" som omger de delar som används under spelet:
        - Bokstavsknappar, för att gissa på bokstäver.
        - Div-element "letterBoxes", där det kommer finnas en ruta per bokstav i det ord som ska gissas.
            - Då spelet startas läggs dessa in som span-element. I CSS-koden utformas de som vita rutor. Då man sedan gissat rätt på en bokstav, skrivs den in i rutan. Se filen style.css.
        - Img-element för att visa bilden med galgen och gubben.
    - Div-element "message" med meddelande till användaren, då spelet är över.



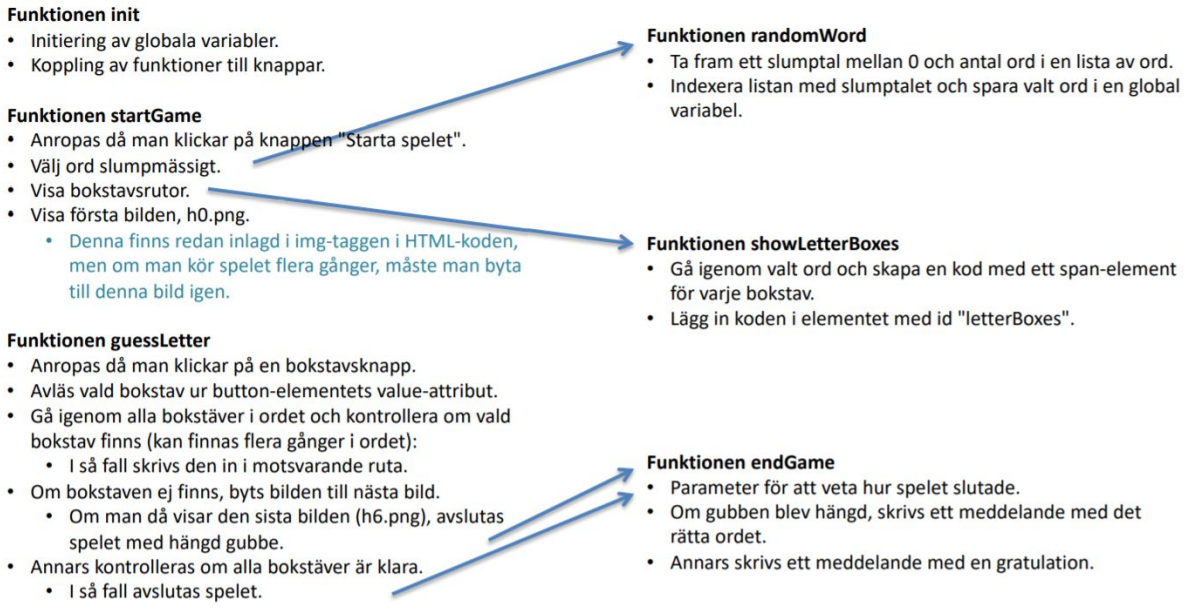
Bildfiler

För att bygga upp bilden med den hängda gubben, ska bilden i img-elementet bytas ut. Följande bilder finns i mappen pics.



På nästa sida finns ett förslag på struktur för uppgiften, men titta inte på den förrän du själv funderat på en struktur för programmet. Diskutera gärna ditt förslag på struktur med en kompis.

Innan man skriver kod, är det ofta bra att tänka igenom vilka funktioner som behövs och skriva korta beskrivningar. Dessa beskrivningar kan man sedan ha som kommentarer i koden, för att förklara funktionerna. När man sedan börjar skriva koden, kanske det tillkommer en del funktioner (man kanske delar upp en funktion i flera), men det underlättar att först ha en genomtänkt planering och struktur. Först skapades listan i den vänstra kolumnen. Det är `init`-funktionen och funktioner som ska anropas, då man klickar på knapparna. Sedan kompletterades det med funktionerna i den högra kolumnen.



Utifrån planeringen med uppdelning i funktioner på föregående sida, tar man fram en lista på vilka globala variabler som behövs. Även här kan det sedan tillkomma flera, men en första planering är alltid bra att göra, innan man börjar skriva koden.

Globala variabler

- wordList – Array med ett antal ord, där man sedan väljer ett slumpmässigt.
- selectedWord – Det ord som valts slumpmässigt och som användaren ska gissa på.
- letterBoxes – Array med referenser till de span-taggar som utgör rutor för bokstäverna i ordet.
- hangmanImg – Referens till img-elementet med bilden för galgen och gubben.
- hangmanImgNr – Nummer för aktuell bil (0-6), för den bildfil som visas (så man sedan kan veta vilket som blir nästa bild).
- msgElem – Referens till div-elementet för meddelanden.

Variabeln `selectedWord` tilldelas ett nytt värde i funktionen `randomWord`. Variabeln `letterBoxes` tilldelas i funktionen `showLetterBoxes`, då span-elementen lagts in i HTML-koden. Variabeln `hangmanImgNr` tilldelas värdet 0, då bilden byts till h0.png i funktionen `startGame`. Övriga variabler initieras i `init`-funktionen.

När vi nu har en struktur med en uppdelning i funktioner och en lista på globala variabler, kan koden börja skrivas.

Globala variabler och init-funktionen

Globala variabler

- I filen script.js lägger du in de globala variabler som finns i listan ovan.
- Glöm inte att också lägga in kommentarerna.

Funktionen `init`

- Lägg in följande lokala variabler:
  - `i` – loopvariabel
  - `startGameBtn` – referens till startknappen
  - `letterButtons` – array med referenser till bokstavsknapparna
- Startknappen
  - Ta fram en referens till knappen "startGameBtn" och spara i variabeln `startGameBtn`.
    - Egentligen behövs ingen variabel endast för att göra nedanstående, men du ska i en senare övning lägga till mer kod för startknappen, så spara referensen i en variabel redan nu.
  - Lägg in kod, så att funktionen `startGame` anropas, då man klickar på knappen.
- Bokstavsknapparna
  - Ta fram en array med referenser till till alla knappar i div-elementet "letterButtons".
  - Gå igenom allihop i en loop och se till att funktionen `guessLetter` anropas då man klickar på dem.
    - Samma funktion ska alltså anropas oavsett vilken av knapparna man klickar på.

```
letterButtons = document.getElementById("letterButtons").getElementsByTagName("button");
for (i=0; i<letterButtons.length; i++) letterButtons[i].onclick = guessLetter;
```

- Bilden och elementet för meddelanden
  - Lägg in en referens till img-elemenet "hangman" i `hangmanImg`.
  - Lägg in en referens till div-elementet "message" i `msgElem`.
- Ordlistan
  - Variabeln `wordList` finns redan med i den fil som du laddat ner till uppgiften, så du slipper skriva in alla ord själv.

Skal till alla funktioner

Nu lägger du in deklaration av de övriga funktionerna. Börja här med att lägga in "skalet" med function, namnet och en kommentar.

Övriga funktioner

- Lägg in skalet för funktionen `startGame`.

```
// Initiera ett nytt spel. Välj ord, visa bokstavsrutor,
// visa första bilden (tom bild) och sätt bildnummer till 0.
function startGame() {

} // End startGame
```

- Lägg på samma sätt in skal för funktionerna:
- `randomWord`
- `showLetterBoxes`
- guessLetter
- `endGame`
  - Denna funktion ska också ha en parameter, som du kan kalla `manHanged`.
  - Då man sedan anropar funktionen skickar man med true eller false, så att man i funktionen kan avgöra vilket meddelande som ska skrivas ut.

Testa

- Öppna nu filen *index.html* i webbläsaren.
- Öppna Webbkonsolen och kontrollera att JS-fel ska visas. Laddasedan om sidan.
- Kontrollera att du inte får några felmeddelanden.
- I övrigt ska inget hända, eftersom funktionerna ännu så länge är tomma.

I tidigare planering beskrevs funktionerna enligt rutorna till höger. Detta ska nu översättas till kod.

Kod i funktionen `startGame`

- Anrop av funktionen `randomWord`.
- Anrop av funktionen `showLetterBoxes`.
- Lägg in bilden "pics/h0.png" i img-elementet som refereras av `hangmanImg`.
- Lägg in 0 i variabeln `hangmanImgNr`.

- Välj ord slumpmässigt.
- Visa bokstavsrutor.
- Visa första bilden, h0.png.
- Variabeln hangmanImgNr tilldelas värdet 0

Kod i funktionen `randomWord`

- Lägg in en lokal variabel som du kallar `wordIndex`.
- Generera ett slumptal mellan 0 och antal ord i `wordList`. Spara slumptalet i `wordIndex`.

- Ta fram ett slumptal mellan 0 och antal ord i en lista av ord.
- Indexera listan med slumptalet och spara valt ord i en global variabel.
- Variabeln selectedWord tilldelas ett nytt värde.

- I `selectedWord` sparar du det ord som indexeras med `wordIndex`.

Kod i funktionen `showLetterBoxes`

- Inför två lokala variabler, som du kallar `newCode` och `i`.
  - `newCode` ska bli en textsträng med HTML-koden för bokstavsrutorna och `i` är en loop-variabel.
- Initiera `newCode` med en tom sträng (två citattecken, utan något mellan dem).
- Lägg in en for-loop, där du går igenom alla tecken i `selectedWord`.
  - För varje varv i loopen lägger du till (+=) strängen " " till `newCode`.
    - Man måste ha med ett nonbreakable space ( ) för att CSS-koden ska funka och en "tom" ruta visas.
- Efter loopen lägger du in `newCode` i elementet med id "*letterBoxes*".
- I variabeln `letterBoxes` tar du fram en array med referenser till alla span-taggar. Börja med *getElementById* och fortsätt sedan med *getElementsByTagName*.

- Gå igenom valt ord och skapa en kod med ett span-element för varje bokstav.
- Lägg in koden i elementet med id "letterBoxes".
- Variabeln letterBoxes tilldelas, då span-elementen lagts in i HTML-koden.

Testa

- Testa programmet i webbläsaren.
- Klicka på knappen "Starta spelet". Du ska då få ett antal rutor under bokstavsknapparna

Funktionen `guessLetter - 1`

- Inför lokala variabler som du kallar `letter`, `i`, `letterFound` och `correctLettersCount`
  - `letter` ska vara bokstaven för knappen, `i` är en loopvariabel, `letterFound` ska användas som en flagga (true/false), för att avgöra om man hittar bokstaven i ordet och `correctLettersCount` ska var en räknare, för att se hur många korrekta bokstäver som hittats.
- Lägg in en rad för att avläsa knappens bokstav ur button-elementet.

- Avläs vald bokstav ur button-elementets value-attribut.
- Gå igenom alla bokstäver i ordet och kontrollera om vald bokstav finns (kan finnas flera gånger i ordet):
  - I så fall skrivs den in i motsvarande ruta.
- Om bokstaven ej finns, byts bilden till nästa bild.
  - Om man då visar den sista bilden (h6.png), avslutas spelet med hängd gubbe.
- Annars kontrolleras om alla bokstäver är klara.
  - I så fall avslutas spelet.

- En referens till den knapp som användaren klickat på finns i this.
- Bokstaven finns i value-attributet, se filen *index.html*.

```
letter = this.value;
```

- Loop för att kontrollera bokstäverna
  - Lägg in en for-loop där du går igenom alla tecken i `selectedWord`.
    - I loopen ska du ha en if-sats där du jämför `letter` med det tecken i `selectedWord` som bestäms av loopvariabeln `i`.
      - Använd `charAt` för att ta ut tecknet ur textsträngen.
    - Om de är lika lägger du in letter i motsvarande textruta, dvs det span-element som bestäms av `letterBoxes` som indexeras med `i`.
      - Glöm inte innerHTML

Testa

- Även om du nu endast gjort de första punkterna i funktionen, är det tillräckligt för att kunna testa.
- Klicka på "Starta spelet".
- Klicka sedan på bokstavsknappar.
- Då bokstaven finns i ordet, ska den skrivas ut i rutorna.

Funktionen `guessLetter - 2`

- För att avgöra om bokstaven fanns i ordet, behövs en "flagga":
  - Före loopen lägger du in false i variabeln `letterFound`.
  - Inuti if-satsen i loopen lägger du in true i `letterFound`.
- Efter loopen lägger du in en ny if-sats där du kontrollerar om `letterFound` är false.
  - I så fall fanns inte bokstaven och nästa bild ska visas.
    - Nästa bild bestäms av `hangmanImgNr`, så räkna först upp den variabeln med 1.
    - Lägg sedan in "pics/h" + hangmanImgNr + ".png" i img-taggen som refereras av `hangmanImg`.
  - Sedan ska du ha ytterligare en if-sats, för att kontrollera om man nu kommit fram till den sista bilden (dvs om `hangmanImgNr` är 6).

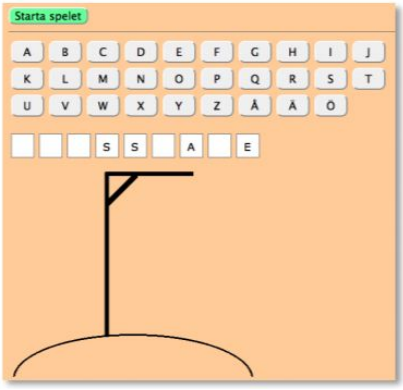
- Avläs vald bokstav ur button-elementets value-attribut.
- Gå igenom alla bokstäver i ordet och kontrollera om vald bokstav finns (kan finnas flera gånger i ordet):
  - I så fall skrivs den in i motsvarande ruta.
- Om bokstaven ej finns, byts bilden till nästa bild.
  - Om man då visar den sista bilden (h6.png), avslutas spelet med hängd gubbe.
- Annars kontrolleras om alla bokstäver är klara.
  - I så fall avslutas spelet.

- Denna if-sats ska alltså ligga inuti den första. Strukturen blir enligt vidstående bild.
- I så fall ska spelet avslutas. Anropa då funktionen `endGame`, med parametern true.

```
if ( ...bokstaven hittades ej... ) {
    ... byt bild ...
    if ( ... sista bilden ...) {
        avsluta med true
    }
}
```

Testa

- Nu kan du testa om bilderna byts ut.
- Klicka på knappen "Starta spelet".
- Klicka sedan på bokstavsknappar
- Varje gång du trycker på en bokstav som inte finns i ordet, ska bilden bytas ut.



Funktionen `guessLetter - 3`

- För att avgöra om alla bokstäver är korrekta, ska du kontrollera om alla bokstavsrutor är ifyllda. Du behöver då en räknare, som du sedan kan jämföra med antalet bokstäver i ordet.
  - Före loopen lägger du in 0 i variabeln `correctLettersCount`.
  - Inuti loopen lägger du in ytterligare en if-sats under den första. I denna if-sats kontrollerar du om rutan ej är tom, dvs om letterBoxes[i].innerHTML är skilt ifrån " ".
    - I så fall innehåller den en bokstav och du räknar upp `correctLettersCount` med 1.

- Avläs vald bokstav ur button-elementets value-attribut.
- Gå igenom alla bokstäver i ordet och kontrollera om vald bokstav finns (kan finnas flera gånger i ordet):
  - I så fall skrivs den in i motsvarande ruta.
- Om bokstaven ej finns, byts bilden till nästa bild.
  - Om man då visar den sista bilden (h6.png), avslutas spelet med hängd gubbe.
- Annars kontrolleras om alla bokstäver är klara.
  - I så fall avslutas spelet.

- Nu kommer vi in på den inringade punkten i rutan ovan.
  - Lägg in en else-if-del till den yttre av if-satserna som ligger under loopen.
    - Strukturen blir alltså nu enligt vidstående bild.

```
if ( ...bokstaven hittades ej... ) {
    ... byt bild ...
    if ( ... sista bilden ...) {
        avsluta med true
    }
}
else if ( ... antal korrekta bokstäver är lika med antal bokstäver i ordet ... ) {
    avsluta  med false
}
```

        - I if-satsen jämför du `correctLettersCount` med antal tecken i `selectedWord`.
            - Är de lika, anropar du funktionen `endGame`, med parametern false.

Testa

- Nu testar du i webbläsaren igen.
- Du får inget meddelande, då spelet är slut, eftersom endGame ännu så länge är tom.
- Men testa programmet ändå, för att se att du inte får några felmeddelanden i debuggern.

Funktionen `endGame`

- Parameter för att veta hur spelet slutade.
- Om gubben blev hängd, skrivs ett meddelande med det rätta ordet.
- Annars skrivs ett meddelande med en gratulation.

Då du skapade skalet till funktionen tidigare i uppgiften, la du också in en parameter kallad `manHanged`. Då du anropat funktionen har du skickat med true eller false till denna parameter. Du ska nu skriva koden i funktionen.

Kod i funktionen endgame

- I en if-sats kontrollerar du om `manHanged` är true.
    - I så fall skriver du i elementet för meddelanden (som du refererar till med msgElem) att gubben hängdes. Skriv också ut vad rätt ord är (dvs variabeln `selectedWord`).
- Annars har du en else-del.
    - I den skriver du ut ett meddelande som gratulerar användaren till att ha klarat av hela ordet.

Testa

- Testa nu i webbläsaren igen.
- Kontrollera att du får ut ovanstående meddelanden, då gubben blir hängd eller då du lyckas gissa alla bokstäver i ordet.

Är programmet klart?

Nu kan man ju klicka på samma bokstav flera gånger. Då man klickat på en bokstav, ska knappen markeras, så att man inte kan klicka på den igen. Det kan göras genom att inaktivera knappen med egenskapen disabled.
Man kan också börja klicka på bokstäverna innan man startat spelet och något ord valts. Eller man kan klicka på knappen för att starta spelet, innan man är klar med det spel som pågår. Så alla dessa knappar behöver också aktiveras och avaktiveras vid en del ställen i programmet.
Ett annat tillägg som ska göras är att i funktionen `randomWord`, se till att inte samma ord väljs två gånger i rad.
Ett tredje tillägg är att mäta tiden, för att se hur lång tid spelet tar.

Aktivera och avaktivera knappar

Förändringar ska ske på följande ställen:

- Då sidan laddats in, alltså i funktionen `init`.
  - Startknappen aktiveras och bokstavsknapparna inaktiveras.
- Då man startar spelet, alltså i funktionen `startGame`.
  - Startknappen inaktiveras och bokstavsknapparna aktiveras.
- Då spelet avslutas, alltså i funktionen `endGame`.
  - Startknappen aktiveras och bokstavsknapparna inaktiveras.
- Då man klickar på en bokstavsknapp, alltså i funktionen `guessLetter`.
  - Den knapp man klickat på ska inaktiveras.

Man ändrar aktivering genom att lägga in *true* eller *false* i egenskapen *disabled*:

```
• knappReferens.disabled = true; // Knappen inaktiveras
• knappReferens.disabled = false; // Knappen aktiveras
```

Globala variabler

För att kunna komma åt referenser till knapparna i de olika funktionerna, måste variablerna `startGameBtn` och `letterButtons` vara globala variabler. De finns nu som lokala variabler i init-funktionen.

- Flytta deklarationen av dessa två variabler från `init` till början av filen, där du har dina globala variabler.

Funktion för att ändra aktivering

Aktivering och inaktivering av knapparna läggs i en separat funktion.

- Skapa en funktion som du kallar `changeButtonActivation`.
- Den ska ha en parameter som du kan kalla `status`.
- I funktionen kontrollerar du status i en if-sats.
  - Om status är true, ska du sätta disabled för `startGameBtn` till false och disabled för alla `letterButtons` till true.
    - Det sistnämnda får du göra i en loop.

- Annars, ska du göra tvärtom, dvs disabled för `startGameBtn` sätts till true och disabled för alla `letterButtons` sätts till false.

Anrop av funktionen changeButtonActivation

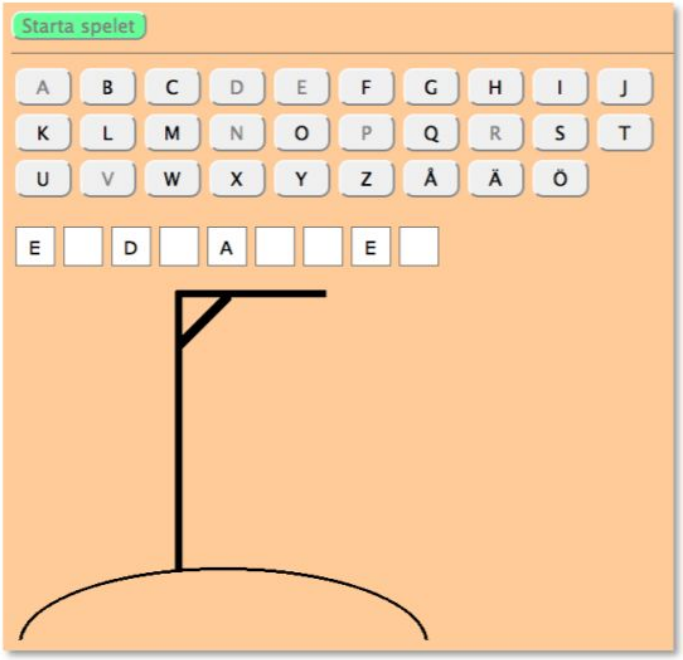
- Sist i init lägger du in `changeButtonActivation(true);`
- Sist i startGame lägger du in `changeButtonActivation(false);`
- Sist i endGame lägger du in `changeButtonActivation(true);`

Ändra bokstavsknapp som man klickar på

I funktionen `guessLetter` har du en referens till knappen i this.
I början av funktionen lägger du in this.disabled = true;

Testa

- Ladda om sidan *index.html* i webbläsaren.
  - Startknappen ska då vara aktiv, men bokstavsknapparna inaktiva. De är "grå" och det händer inget om du klickar på dem.
- Klicka på knappen "Starta spelet".
  - Då inaktiveras startknappen, men alla bokstavsknappar blir aktiva.
- Klicka på knappar för bokstäver.
  - Den knapp du klickar på blir inaktiv.
- Gå igenom spelet tills gubben blir hängd eller du fått fram rätt ord.
  - Startknappen blir då aktiv och bokstavsknapparna blir inaktiva.



Ta bort gammalt meddelande

Då man startar spelet igen, ska det gamla meddelandet från föregående spel tas bort.

Funktionen `startGame`

- Sist i funktionen lägger du till en programsats, där du lägger in en tom sträng i elementet för meddelanden.

Testa

- Kör igenom spelet och klicka på startknappen igen.
- Då tas meddelandet längst ner på sidan bort.

Välj ej samma ord två gånger i rad

Då man startar spelet igen, ska inte samma ord som förra gången kunna väljas igen. Detta sker i funktionen `randomWord`, som nu ska modifieras lite.

Förändringar i koden i funktionen randomWord

- Inför ytterligare en variabel som du kallar `oldWord`.
- Efter variabeldeklarationerna lägger du in en rad, där du sparar `selectedWord` i `oldWord`.
- De två programsatser som du redan har i funktionen, lägger du i en while-loop.
  - Villkoret i loopen ska vara att `oldWord` är lika med `selectedWord`
  - Man går alltså runt i loopen och genererar ett nytt slumptal och ord, så länge det är likadant som det gamla ordet. Men då man får ett annat ord, lämnas loopen.

Testa

Testa i webbläsaren och kontrollera att du inte får något felmeddelande i debuggern.

Mäta tiden för spelet

För tidsmätningen kommer funktionen *getTime* i *Date*-objektet användas. Den ger tiden i millisekunder från den 1 januari 1970. Detta har dock ingen betydelse här, utan vi kommer ta fram *getTime* vid två olika tillfällen och sedan räkna ut skillnaden mellan dem.

Det sista tillägget du ska göra är att mäta hur lång tid det tar för användaren att gissa ordet eller få gubben hängd.
Detta görs genom att vi först tar fram tiden då spelet startas och sedan tiden då spelet avslutas. Skillnaden blir då speltiden. Tiden räknas ut i funktionen `endGame`, men starttiden måste vi spara i funktionen `startGame`. Det måste då läggas i en global variabel.

Globala variabler

Inför en global variabel som du kallar `startTime`.

Funktionen startGame

- Deklarera en variabel som du kallar `now`.
- I slutet av funktionen lägger du till kod för att skapa ett nytt Date-objekt och sedan avläsa `getTime` och spara i `startTime`.

```
now = new Date();
startTime = now.getTime();
```

Funktionen endGame

- Deklarera två variabler som du kallar `now` och `runTime`.
- I början av funktionen lägger du in kod för att skapa ett nytt Date-objekt och sedan beräkna skillnaden mellan `getTime` i objektet och `startTime`. Tiden blir i millisekunder, så dividera med 1000, för att få det i sekunder.

```
now = new Date();
runTime = (now.getTime() - startTime) / 1000;
```

- I slutet av funktionen lägger du till en rad där du skriver ut `runTime` i elementet för meddelanden.
- Använd toFixed(1), för att avrunda till en decimal.

```
runTime.toFixed(1)
```

Testa

- Testa i webbläsaren och kontrollera att du får ut tiden.