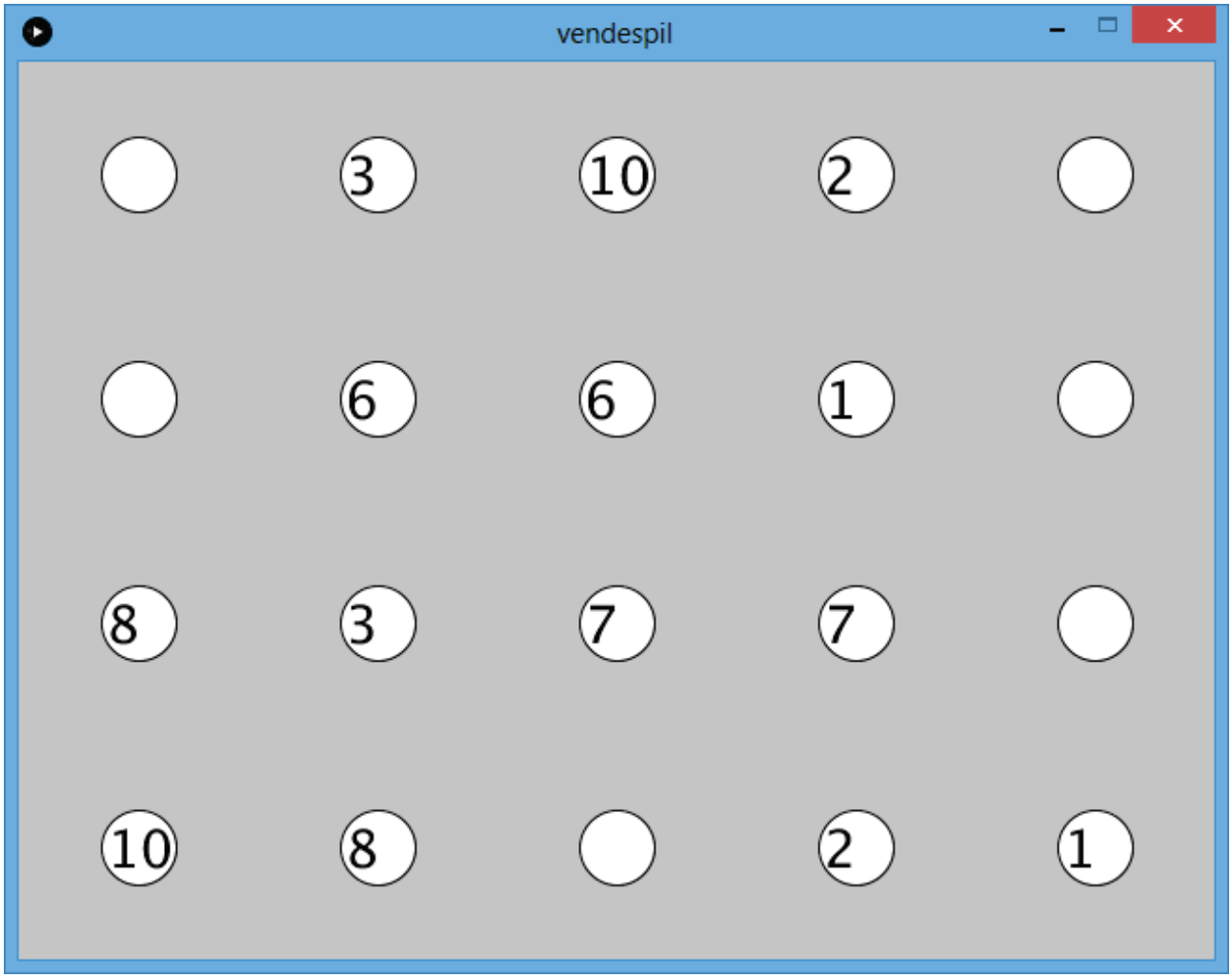
# Vendespil

Programmering C




Bent Arnoldsen - Lærer
Eksempel dokumentation
11/11-2015
Holstebro HTX

# Indholdsfortegnelse

# Indledning

Denne rapport er udformet som et eksempel på en dokumentation af et programmerings-projekt, og er lavet for at inspirere til hvordan software dokumentation kan udformes.

Dokumentationen skal blot tages som et eksempel, og ikke som en skabelon over hvad der skal være med i dokumentationen.

Projektet tager udgangspunkt i undervisningen i to-dimensionelle arrays.

# Problemformulering

Programmet skal kunne illustrere et simpelt men funktionelt vendespil. Der lægges ikke vægt på det grafiske udseende men funktionaliteten i vendespillet.

## Kravspecifikation

Vendspillet skal kodes, så man ved at rette i koden kan bestemme antallet af brikker i spillet på simpel vis.

Brikkerne skal bestemmes ved tal, hvor der skal være skal være to af hvert tal i spillepladen. Brikkerne skal placeres tilfældigt.

Ved første klik skal brikken ”vendes”, så man kan se det tal man har klikket på.

Ved næste klik skal den næste brik også ”vendes”, med mindre det er en brik man har klikket på før.

- Hvis de to brikker er ens, så forbliver de vist og der startes ved første klik igen.
- Hvis brikkerne ikke er ens, så skal de vises et stykke tid, så man har mulighed for at memorere placeringen af de to forskellige brikker

Når alle brikker er vist, så skiftes til en vinder-visning.

## Udviklingsstrategi

Programmet er udviklet som en forlængelse af undervisningen i to-dimensionelle arrays, og derfor tager koden udgangspunkt i den kode der blev anvendt til den undervisning.

Koden der er startet i er vist i bilag 2.

For at komme frem til det resultat der er afleveret, er der anvendt stepwise improvement (UC Holstebro 2013), hvor det er sket primært i følgende step:

- Fjerne optællingen i felterne og lægge to ind af hvert tal (par der skal findes) – der ventes med at blande dem, da det er lettere at teste, når de ikke er blandet
- Få programmet til at registrere første klik og andet klik, hvor man så kan se om de er ens.
- Få etableret så der kun vises dem der er ens, og dem man har klikket på.
- Få lavet så de to der er klikket på vises i 3 sekunder og forsvinder hvis de er forskellige
- Få lavet så man kan klikke på nye forsøg inden de 3 sekunder er gået, og at de viste forsvinder
- Få etableret en Game-over skærm, når man har fundet alle par
- Etablere blanding af tallene i spillet, så det bliver et reelt vendespil.

Hver enkelt step er testet under og som afslutning på udviklingen

# Programdokumentation

Selve programmet dokumenteres på forskellige niveauer.

## Data forklaringer

I dette afsnit forklares hvad de forskellige variabler kan indeholde og hvad det betyder, altså hvilke dataabstraktioner programmet indeholder.

De to første variabler kan betragtes som konstanter, da de ikke ændres i koden – disse to variabler angiver hvor mange brikker der er hhv. vandret og lodret.

```
int bredde = 5;
int hoejde = 4;
```

Næste variabel angiver ud fra højde og bredde hvor mange brik-par der er. Denne variabel kan også betragtes som en konstant.

```
int max = bredde * hoejde / 2;
```

Variablen antal angiver hvor mange par der er fundet. Denne variabel starter på 0.

```
int antal = 0;
```

Det todimensionelle array brik indeholder numrene der vises på alle brikker. Hvis der er 20 brikker i alt, så indeholder brikkerne tallene fra 1 til 10 begge inklusive, hvert af tallene to gange.

```
int[][] brik = new int[bredde][hoejde];
```

Et tilsvarende todimensionelt array vis angiver hvordan hver af de aktuelle brikker skal vises. Indholdet i vis-arrayet skal tolkes som følger:

- 0: brikken skal ikke vises
- 1: brikken har midlertidig visning
- 2: brikken er fundet sammen med den anden brik, og skal blive ved med at blive vist

```
int[][] vis = new int[bredde][hoejde];
```

Variablen first angiver hvilken værdi den først klikkede brik har. Hvis værdien er 0, så er der ikke klikket på noget, ellers vil den være fra 1 til max.

```
int first = 0;
```

Variablerne lastX, lastY, lastX2 og lastY2 bruges til at huske hhv. første og andet klik, så man efterfølgende kan nulstille visningen. Indholdet er indexet til arrayet (X- og Y-værdi fordi det er todimensionelt).

```
int lastX, lastY, lastX2, lastY2;
```

Variablen show bruges til at holde styr på hvor lang tid der er tilbage. Den sættes til 90, hvilket svarer til 1½ sekund, fordi skærmen opdateres 60 gange i sekundet.

```
int show = 0;
```

# Starten af programmet

Det første der skal ske i programmet er at data gøres klar, så man kan spille spillet. Dette gøres kun en gang, så det laves i setup().

## Flowcharts

Det første flowchart i Figur 1 angiver hvordan programmets data initialiseres, så der kommer en spilleplade ud af det. Gennemløbet af spillepladen laves som to forløkker inden i hinanden.


*Figur 1 Flowchart for Initialiseringen af spillet*

## Kodeforklaring

Her forklares hvordan spillepladen bliver blandet, så spillet starter tilfældigt hver gang.

```
// Bland spillepladen ved at bytte to tilfældige felter mange gange
for (int n = 0; n < 100; n++) {
  int x = int(random(bredde));
  int y = int(random(hoejde));
  nr = brik[x][y];
  int x2 = int(random(bredde));
  int y2 = int(random(hoejde));
  brik[x][y] = brik[x2][y2];
  brik[x2][y2] = nr;
}
```

Der laves en forløkke, som bliver gennemløbet 100 gange. Dette er måske ikke helt nok ved store spilleplader, men til en 4 gange 5 spilleplade er det OK.

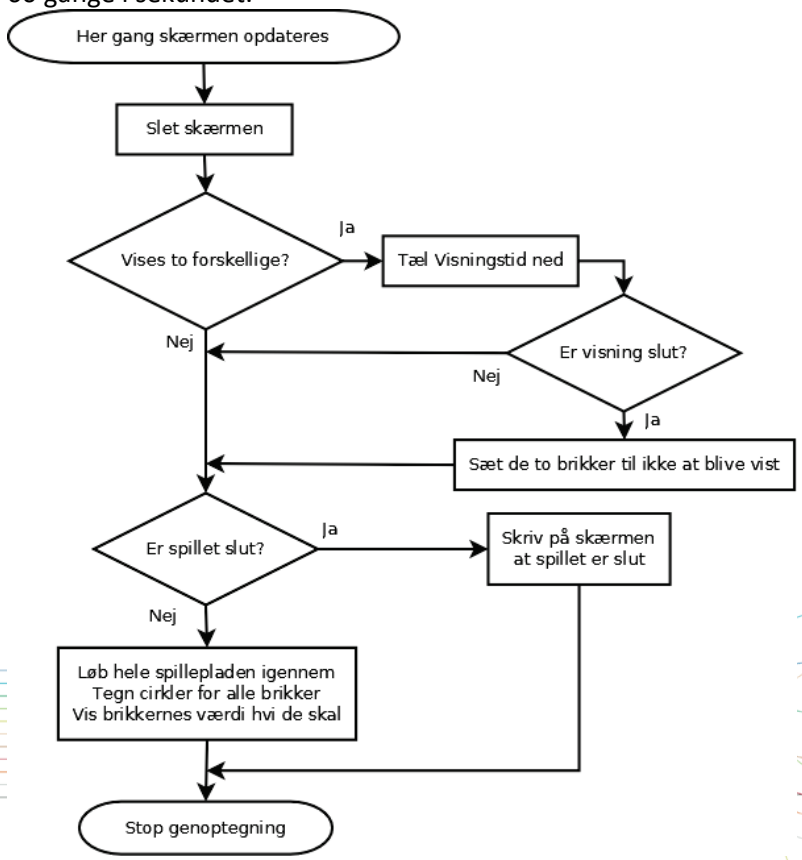
Der findes en tilfældig brik ved at lave en x og en y position i brik-arrayet. Denne brik hentes ud i variablen nr. Der findes en anden tilfældig brik ved en x2, y2 position i brik-arrayet. Denne brik placeres i position x, y hvorefter brikken i variablen nr lægges ind i position x2, y2. På denne måde byttes de to positioner. Det kan med lav sandsynlighed ske at det er samme brik der byttes med, men dette vil ikke skade noget.

## Test af blandefunktionen

Blandefunktionen er blevet testet ved at programmet sættes til at vise alt indholdet, og ved at starte programmet 10-20 gange er det vurderet at der hver gang er lavet en passende blanding af brikkerne.

# Visning på skærmen

Flowchartet i Figur 2 viser hvordan skærmen opdateres – opdateringen sker med standard opdateringen på 60 gange i sekundet.




*Figur 2 Flowchart der illustrerer hvordan skærmen opdateres i draw-funktionen*

## Kodeforklaring

Koden der tegner cirklerne (illustration af spillebrikkerne) og de tal der skal vises laves ved hjælp af to forløkker inden i hinanden, der løber alle brikker igennem, som vist i den følgende kode:

```
for (int n = 0; n < bredde; n++) {
  for (int i = 0; i < hoejde; i++) {
    // Tegn Cirkler
    fill(255);
    ellipse(n * width/bredde + width/bredde/2, i * height/hoejde + height/hoejde/2, 40, 40);
    // vis de brikker der står til visning
    fill(0);
    textSize(28);
    if (vis[n][i] > 0) {
      text(brik[n][i], n * width/bredde + width/bredde/2 - 17, i * height/hoejde + height/hoejde/2+11);
    }
  }
}
```

Alle cirker tegnes med hvis fyldfarve i en position der beregnes ud fra brikkens position i arrayet samt højde og bredde af programvinduet.
Med sort tekst skrives indholdet af brikken, hvis den skal vises. Her betyde 0 at den ikke skal vises, 1 betyder at det er en midlertidig visning (at der er klikket på brikken) og 2 betyder at parret er fundet.

Mekanismen i den midlertidige visning er forklaret i næste afsnit.

## Test af visningen

Visningen er blevet bruget gennem hele udviklingen af programmet, så den er gennemtestet på denne måde. Visningen fungerer tilfredsstillende. En ændring kunne være at encifrede tal centreres.

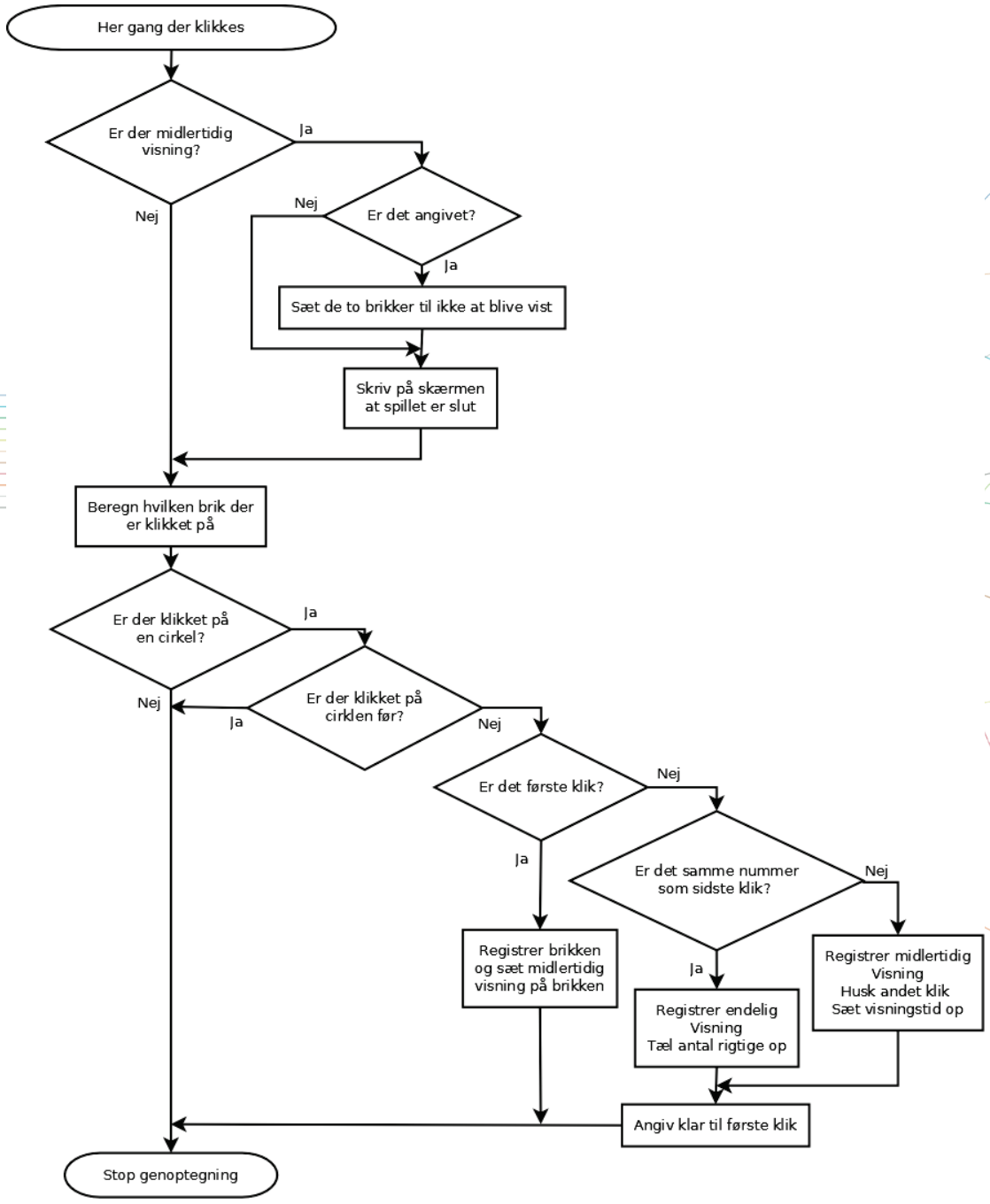
# Registrering af museklik

Når der registreres et museklik på skærmen, så skal dette klik omsættes til en registrering af hvilken brik der er klikket på og hvilket klik det er i sekvensen af at få vist de to brikker.

Visningens opførsel hænger sammen med klikket, så noget kode forklares sammen med klikket.

## Flowchart over museklik

Flowchartet der ses i Figur 3 angiver hvad der sker når der klikkes på skærmen.




*Figur 3 Flowchart der illustrerer et museklik*

## Kodeforklaring for klik på brikkerne

For at registrere hvilken cirkel der rammes, så beregnes hvilket x- og y-index klikket svarer til. Dette gøres i følgende kode:

```
// Find indexerne på den nærmeste cirkel
int x = mouseX*bredde/width;
int y = mouseY*hoejde/height;
// Begræns klikket til et kvadrat rundt om cirklen
if (mouseX > x * width/bredde + width/bredde/2 - 20) {
  if (mouseX < x * width/bredde + width/bredde/2 + 20) {
    if (mouseY > y * height/hoejde + height/hoejde/2 - 20) {
      if (mouseY < y * height/hoejde + height/hoejde/2 + 20) {
```

Da der er luft mellem cirklerne så bestemmes et kvadrat rundt om cirklen til at være et klik på en cirkel, dette er lavet med de viste 4 if-sætninger, som alle skal være opfyldt for at være inde i kvadratet.

Hvis der er klikket på en cirkel har vi i x, y registreret hvilken cirkel der er klikket på, der udføres så følgende kode:

```
if (vis[x][y] == 0) {
  if (first == 0) {  // Husk hvor der klikkes hvis det er første klik i sekvensen
    lastX = x;
    lastY = y;
    first = brik[x][y];
    vis[x][y] = 1;
  } else {
    if (brik[x][y] == first) { // Hvis der i andet klik klikkes på en med samme nummer
      vis[x][y] = 2;           // Sæt til permanent visning for begge brikker
      vis[lastX][lastY] = 2;
      antal++;
    } else {           // Hvis det er forskellige brikker ved andet klik
      vis[x][y] = 1;   // Sæt til midlertidig visning i 90 frames (ca. 1½ sekund)
      lastX2 = x;
      lastY2 = y;
      show = 90;
    }
    first = 0;  // gør klar til et første klik
  }
}
```

Hvis brikken ikke er vist, så skal der gøres noget – alle viste brikker ignoreres, også første klik i at parre brikkerne (her er der registreret 1 i vis-arrayet på brikpositionen).

Variablen first angiver indholdet af den sidste brik der er klikket på (kun i første klik). Hvis den står på 0, så er det fordi der ikke er klikket første gang. Så hvis det er første klik, så registreres positionen af brikken og brikkens indhold. Brikken sættes til at blive vist ved at sætte vis af dens position til 1.

Hvis det er klik på den anden brik i visnings-sekvensen, så tjekkes der om det er samme indhold som ved første klik. Hvis det er samme indhold, så sættes begge brikker til 2 i vis-arrayet og der registreres at der er fundet et par mere.

Hvis de to brikker har forskelligt indhold, så skal der stadig ske en visning af dem, men den skal slukkes igen. Begge brikker bliver sat til 1 i vis-arrayet, og positionen af dem begge huskes. Visningen sker ved hjælp af variablen show, der sættes til 90. Denne variabel er baseret på at programmet vises med en framerate på 60, så visningen vil ske i 90 frames altså 1½ sekund. Fjernelsen af visningen sker i draw() med følgende kode:

```
if (show > 0) {
  show--;
  if (show == 0) {
    if (vis[lastX][lastY] == 1) {
      vis[lastX][lastY] = 0;
      vis[lastX2][lastY2] = 0;
    }
  }
}
```

Her tælles show en ned hver gang skærmen opdateres indtil show rammer 0, og når den gør det, så sættes visningen af de to brikker til 0 igen.

Hvis brugeren er hurtig, og får klikket på en ny brik inden de 1½ sekund er gået, så vil det gå galt, at visningen ikke bliver fjernet i draw. Derfor indledes mousePressed med følgende kode:

```
if (show > 0) {
  if (vis[lastX][lastY] == 1) {
    vis[lastX][lastY] = 0;
    vis[lastX2][lastY2] = 0;
  }
  show = 0;
}
```

Her fanges det om der er en visning i gang, og hvis der er det, så sættes visningen til 0 af de to sidst klikkede brikker, inden der registreres hvad der er klikket på – dette gør at man også kan klikke på en af de brikker der lige har været klikket på.

### Test af klik på brikkerne.

Denne test er forklaret i sluttesten

## Test af programmet

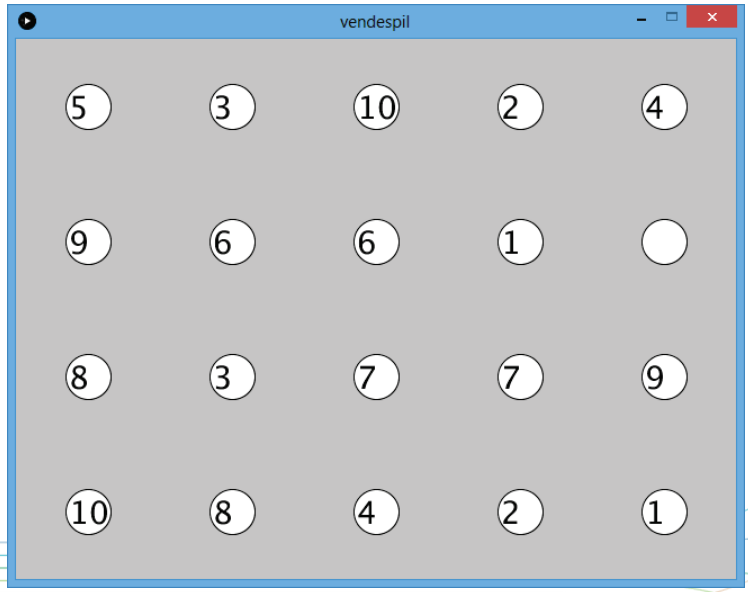
Programmet er naturligvis blevet testet løbende gennem udviklingen af programmet, men for at sikre at alt virker, så er der foretaget en sluttest af programmet. Denne sluttest har aftestet følgende:

Ved det første klik bliver brikken "vendt", så man kan se det tal man har klikket på. Der er kontrolleret at tallet ikke forsvinder efter et stykke tid, ved at vente ca. 10 sekunder.

Ved et klik på næste tal er det testet at den klikkede brik bliver "vendt" og at begge brikker vises i ca. 1½ minut. Det er ligeledes testet at begge brikker forsvinder, hvis der klikkes igen på pladen inden for det 1½ minut, og at det nye klik kan give en ny visning.

Spillet er testet med 3 x 2 brikker, 5 x 4 brikker og 8 x 6 brikker. Alle størrelser fungerer. Højden kan ikke være ulige af hensyn til den måde der bestemmes at der er indlagt 2 ens tal.

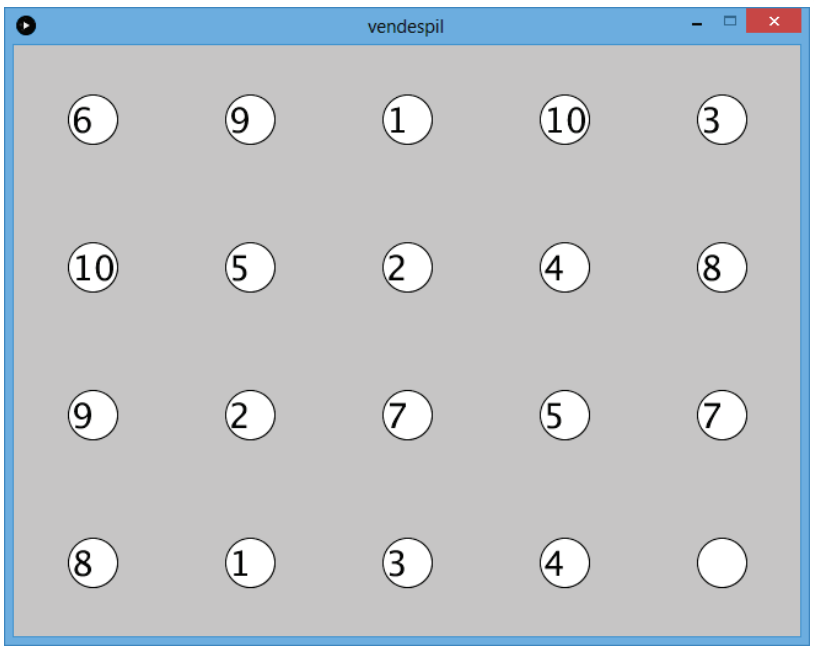
Der er testet at alle brikker er der to gange ved at gennemføre spillet, og som illustreret i Figur 4. Alle tal er vis 2 gange pånær 5-tallet, der ligger bag det blanke felt.




*Figur 4 Spillet testet hvor der kun mangler sidste klik*

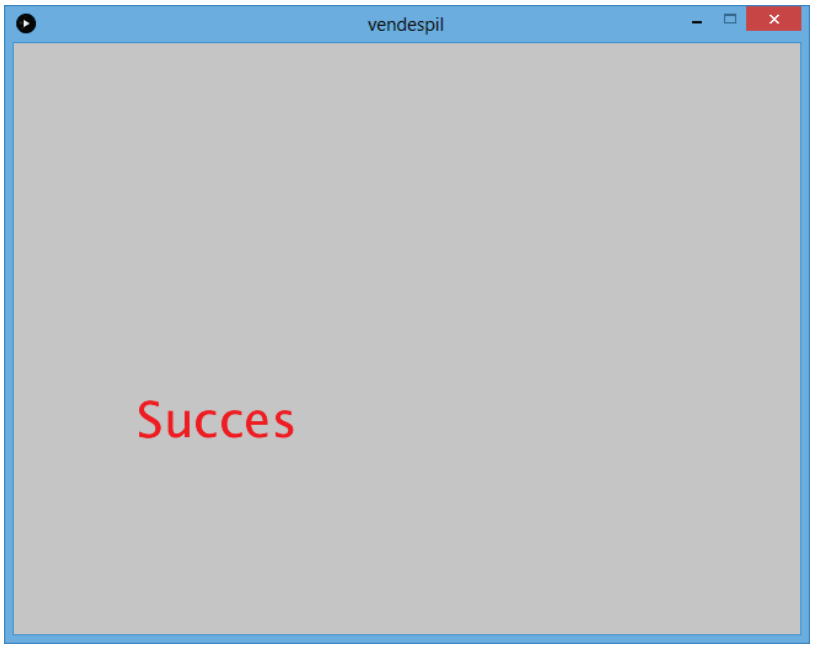
At det er 5-tallet kan ses ved visningen i Figur 6.

For at eftervise at det er tilfældige placeringer gennemføres spillet en gang til (kræver genstart af programmet). Der ses at man kommer frem til en fordeling af tallene som vist i Figur 5.




*Figur 5 Fordeling af tallene ved ny kørsel af programmet*

Ved afslutning af spillet fremkommer hver gang visningen i Figur 6.




*Figur 6 Visning når spillet er gennemført*

## Koden

Selve koden er vedlagt i Bilag 1, hvor de forskellige deles funktion er kommenteret.

## Konklusion

Efter at have dokumenteret og testet programmet, så kan det konkluderes programmet har opnået de ønskede funktioner, som det er angivet i kravspecifikationen, men at der godt kan forbedres nogle ting som det bliver angivet i perspektiveringen.

## Perspektivering

Efterhånden som man tester, så kommer der nye ideer op til hvilke funktioner, det kunne være rart at have i programmet.

- Når man gennemfører, så ville det være smart at man kunne genstarte spillet, så man starter forfra med en ny fordeling.
- Det ville også gøre spillet sjovere, hvis man havde lidt flere registreringer vist på skærmen mens man spiller. Det kunne være tid og antal klik man har brugt på at gennemføre.
- Rent grafisk lader spillet også meget tilbage at ønske, så det kunne være at man får vist billeder i stedet for tal.

# Bilag 1 – Den endelige kode

```
// banens dimensioner, der giver antallet af brikker
int bredde = 5;
int hoejde = 4;  // Skal være et lige tal
int max = bredde * hoejde / 2;
// Hvor mange der er fundet
int antal = 0;
// Indholdet af brikkerne og visnings-status
int[][] brik = new int[bredde][hoejde];
int[][] vis = new int[bredde][hoejde];
// Variabler der registrerer hvor man er i klik-status
int first = 0;
int lastX, lastY, lastX2, lastY2;
int show = 0;

void setup() {
  size(640, 480);
  // Angiv brikkernes værdier og status til at de ikke vises
  int nr = 1;
  for (int n = 0; n < bredde; n++) {
    for (int i = 0; i < hoejde; i++) {
      brik[n][i] = nr;
      if (i % 2 == 1) nr++;
      vis[n][i] = 0;
    }
  }
  // Bland spillepladen ved at bytte to tilfældige felter mange gange
  for (int n = 0; n < 100; n++) {
    int x = int(random(bredde));
    int y = int(random(hoejde));
    nr = brik[x][y];
    int x2 = int(random(bredde));
    int y2 = int(random(hoejde));
    brik[x][y] = brik[x2][y2];
    brik[x2][y2] = nr;
  }
}

void draw() {
  background(200);
  // Hvis der er brikker der skal vise, så tæl visningstiden ned og fjern dem,
  // hvis der ikke skal vises længere
  if (show > 0) {
    show--;
    if (show == 0) {
      if (vis[lastX][lastY] == 1) {
        vis[lastX][lastY] = 0;
        vis[lastX2][lastY2] = 0;
      }
    }
  }
  // Stop visningen hvis spillet er færdigt
  if (antal == max) {
    fill(255, 0, 0);
    textSize(40);
    text("Succes", 100, 320);
  } else {
    for (int n = 0; n < bredde; n++) {
      for (int i = 0; i < hoejde; i++) {
        // Tegn Cirkler
        fill(255);
        ellipse(n * width/bredde + width/bredde/2, i * height/hoejde + height/hoejde/2, 40, 40);
        // vis de brikker der står til visning
        fill(0);
        textSize(28);
        if (vis[n][i] > 0) {
          text(brik[n][i], n * width/bredde + width/bredde/2 - 17, i * height/hoejde + height/hoejde/2+11);
        }
      }
    }
  }
}
```

```
// Funktionen fanger et museklik
void mousePressed() {
  // Hvis der klikkes men der vises, så stoppes visningen
  if (show > 0) {
    if (vis[lastX][lastY] == 1) {
      vis[lastX][lastY] = 0;
      vis[lastX2][lastY2] = 0;
    }
    show = 0;
  }
  // Find indexerne på den nærmeste cirkel
  int x = mouseX*bredde/width;
  int y = mouseY*hoejde/height;
  // Begræns klikket til et kvadrat rundt om cirklen
  if (mouseX > x * width/bredde + width/bredde/2 - 20) {
    if (mouseX < x * width/bredde + width/bredde/2 + 20) {
      if (mouseY > y * height/hoejde + height/hoejde/2 - 20) {
        if (mouseY < y * height/hoejde + height/hoejde/2 + 20) {
          // Hvis der klikkes på en cirkel der ikke vises
          if (vis[x][y] == 0) {
            if (first == 0) {  // Husk hvor der klikkes hvis det er første klik i sekvensen
              lastX = x;
              lastY = y;
              first = brik[x][y];
              vis[x][y] = 1;
            } else {
              if (brik[x][y] == first) { // Hvis der i andet klik klikkes på en med samme nummer
                vis[x][y] = 2;           // Sæt til permanent visning for begge brikker
                vis[lastX][lastY] = 2;
                antal++;
              } else {           // Hvis det er forskellige brikker ved andet klik
                vis[x][y] = 1;   // Sæt til midlertidig visning i 90 frames (ca. 1½ sekund)
                lastX2 = x;
                lastY2 = y;
                show = 90;
              }
              first = 0;  // gør klar til et første klik
            }
          }
        }
      }
    }
  }
}
```

# Bilag 2 – Koden der er startet med

```
int bredde = 8;
int hoejde = 6;
int[][] click = new int[bredde][hoejde];

void setup() {
  size(640, 480);
  // Nulstil anatallet af modtagne klik
  for (int n = 0; n < bredde; n++) {
    for (int i = 0; i < hoejde; i++) {
      click[n][i] = 0;
    }
  }
}

void draw() {
  background(200);
  for (int n = 0; n < bredde; n++) {
    for (int i = 0; i < hoejde; i++) {
      // Tegn Cirkler
      fill(255);
      ellipse(n * width/bredde + width/bredde/2, i * height/hoejde + height/hoejde/2, 40, 40);
      // Marker element indexer
      fill(0);
      String txt = n + " " + i;
      textSize(14);
      text(txt, n * width/bredde + width/bredde/2, i * height/hoejde + height/hoejde/2-22);
      // Angiv antallet af klik i hvert element
      textSize(28);
      text(click[n][i], n * width/bredde + width/bredde/2 - 17, i * height/hoejde + height/hoejde/2+11);
    }
  }
}

void mousePressed() {
  // Find indexerne på den nærmeste cirkel
  int x = mouseX*bredde/width;
  int y = mouseY*hoejde/height;
  // Begræns klikket til et kvadrat rundt om cirklen
  if (mouseX > x * width/bredde + width/bredde/2 - 20) {
    if (mouseX < x * width/bredde + width/bredde/2 + 20) {
      if (mouseY > y * height/hoejde + height/hoejde/2 - 20) {
        if (mouseY < y * height/hoejde + height/hoejde/2 + 20) {
          // Forøg antallet af klik på den aktuelle cirkel
          click[x][y]++;
        }
      }
    }
  }
}
```