Kapittel 22

Prosjekt: pong

Hva er et prosjekt?

I denne boka har vi lagt inn fire *prosjekter*. De fungerer som repetisjon av teoristoffet som har blitt gjennomgått i forkant. Hensikten med prosjektene er å vise hvordan det vi har lært, kan brukes i praksis, og hvordan vi kan *planlegge* applikasjoner og *dokumentere* koden vår med god kommentering.

Prosjektene har ingen oppgaver underveis; hensikten er at du skal gjenskape det som lages i kapitlet. Bakerst i hvert av prosjektkapitlene vil du også finne forslag til hvordan du kan utvide prosjektene.

På elevnettstedet finner du koder og eventuelle filer vi bruker.

P.S. Ikke la deg skremme av mengden kode. Koden kan deles opp i mange biter, som hver for seg er lette å forstå.

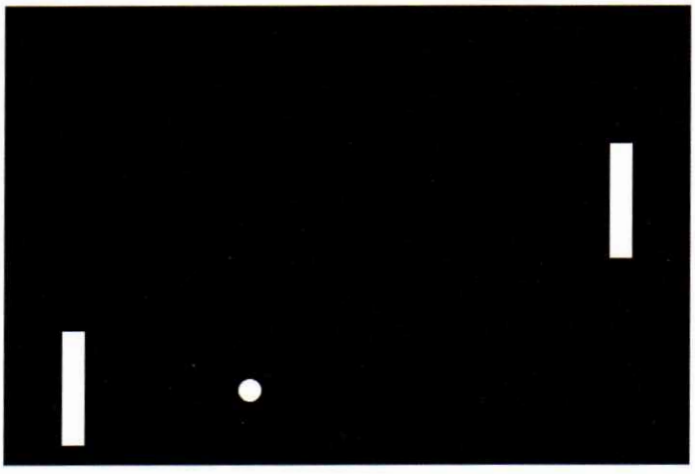
pssst

Husk nettstedet på Lokus.

I vårt siste prosjekt skal vi se på en klassiker, nemlig «pong», en av de første store spillsuksessene. Spillet er enkelt, men likevel morsomt, og vi får sett på en del sentrale konsepter innen spillprogrammering. I figur 22.1 på neste side kan du se hvordan vårt resultat ser ut. Du kan selvfølgelig tilpasse prosjektet slik at det passer deg, for eksempel ved å velge andre farger.

22.1 Planlegging

For å planlegge pong skal vi bruke en kravspesifikasjon og pseudokode som tidligere, men vi skal også se på et nytt planleggingsverktøy — UML-diagrammer.



Figur 22.1: Skjermbilde av spillet.

Kravspesifikasjon

En enkel kravspesifikasjon for pong kan se slik ut:

- Spillet skal ha én ball som spretter rundt i en firkantet bane, og to spillere, representert ved høye rektangler plassert på venstre og høyre side av banen.
- Ballen skal sprette når den møter toppen eller bunnen av banen, men den skal forlate banen på venstre og høyre side.
- Når ballen forlater banen, skal den starte på nytt i midten av banen med en tilfeldig retning.
- Spillerne (rektanglene) skal kunne styres opp og ned av to brukere, og ballen skal sprette vekk når den treffer dem.
- Ballen skal overta spillerens fart i y-retning. Det vil si at hvis spilleren er på vei opp når ballen treffes, så skal ballen få økt sin fart oppover i y-retning.

Det finnes også andre naturlige krav for dette prosjektet, men for å gjøre koden i kapitlet oversiktlig har vi valgt å fokusere på de viktigste kravene. På side 331 kan du se forslag til andre ting som kan legges til i prosjektet.

Pseudokode

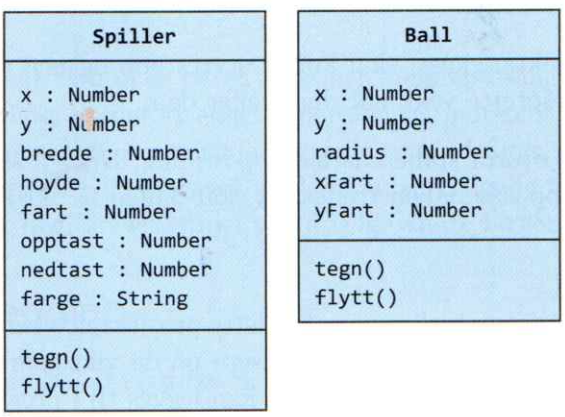
Dette spillet byr ikke på noen avanserte algoritmer som er vanskelige å løse, men det er viktig å ha en god oversikt over gangen i spillet. Vi skal derfor lage en kort pseudokode som beskriver «spillmotoren» vår, altså den funksjonen vi skal gjenta ved å bruke `requestAnimationFrame()`.

```
FUNKSJON spill
  tøm alt innholdet i canvas-elementet

  flytt spiller 1
  tegn opp spiller 1
  flytt spiller 2
  tegn opp spiller 2

  flytt ballen (og sjekk om den kolliderer med noe)
  tegn opp ballen
```

Når vi vet hva vi ønsker at «spillmotoren» vår skal utføre, vet vi også hvilke objekter og metoder vi har behov for. Disse oppsummerer vi med UML-diagrammet nedenfor.

UML-diagram

UML står for «Unified Modeling Language», og er ment som et standardisert planleggings-«språk» for programvareutvikling. UML omfatter mye, men vi skal se på det som kalles et *klassediagram*. I figur 22.2 nedenfor kan du se klassediagrammet vi bruker for å planlegge pong. Vi angir klassenes navn øverst, deretter angir vi egenskaper i midten og metoder nederst. Ved å lage et klassediagram som tar hensyn til alle egenskaper og metoder vi ønsker oss, blir det lettere å lage klassene med JavaScript senere.



Figur 22.2: Et UML-diagram lar oss planlegge klassene vi ønsker å lage. Her er klassene `Spiller` og `Ball` illustrert med sine egenskaper og metoder.

22.2 Koding

HTML og CSS

I dette prosjektet er HTML- og CSS-koden veldig enkel. Vi lager et `<canvas>`-element som vi midtstiller og gir en bakgrunnsfarge. Du kan se utgangspunktet for koden vår i kode 22.1 nedenfor.

```
<!doctype html>
<html>
<head>
  <title>PONG</title>
<style>
  body {
    background-color: #323232;
  }
  #mittCanvas {
    background-color: black;
    display: block;
    margin: auto;
  }
</style>
</head>
<body>

<canvas id="mittCanvas" width="600" height="400"></canvas>

<script>
  /* Her skal vi skrive JavaScript-koden */
</script>

</body>
</html>
```

Kode 22.1: HTML-koden vi bruker i pong.

JavaScript

Før vi ser på detaljene i koden vår, skal vi se på «spillmotoren» vår (kode 22.2 på neste side). All annen kode vi skal skrive, blir brukt i denne «motoren». Vi starter med å tømme alt innholdet i `<canvas>`-elementet, før vi flytter og tegner opp de to spillerne. Til slutt flytter vi ballen og tegner den opp før vi gjentar hele prosessen.

Denne koden plasserer vi helt nederst i `<script>`-elementet.

```
// Funksjonen spill() kjører selve spillet,
// og gjentas i det uendelige
function spill() {
  // Tømmer alt innhold på canvas-elementet
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Flytter og tegner opp de to spillerne
  spiller1.flytt();
  spiller1.tegn();
  spiller2.flytt();
  spiller2.tegn();

  // Flytter og tegner opp ballen
  ball.flytt(spiller1, spiller2);
  ball.tegn();

  // Gjentar funksjonen spill()
  requestAnimationFrame(spill);
}

// Setter i gang funksjonen spill() første gang
requestAnimationFrame(spill);
```

Kode 22.2: «Spillmotoren» (animasjonsfunksjonen) vi bruker i pong.

Nå kan vi begynne på oppsettet vårt. I kode 22.3 nedenfor henter vi `<canvas>`-elementet og lager et oppsett for tastaturhendelser. Denne koden ble forklart i forrige kapittel, så vi går ikke nærmere inn på den her.

```
"use strict"

// Henter canvas-elementet
var canvas = document.querySelector("#mittCanvas");
var ctx = canvas.getContext("2d");

// Lager en tom array som skal holde på tastene som trykkes ned
var taster = [];

// Lytter etter keydown- og keyup-hendelser
window.addEventListener("keydown", knappnedopp);
window.addEventListener("keyup", knappnedopp);

// Når en knapp holdes nede,
// registreres den som true i arrayen taster,
// tilsvarende slettes den (settes til false)
// når den ikke lenger holdes nede
function knappnedopp(e) {
  if (e.type === "keydown") {
    taster[e.keyCode] = true;
  } else if (e.type === "keyup") {
    delete taster[e.keyCode];
  }
}
```

Kode 22.3: Oppsett for tastaturhendelser.

Klassen `Spiller`, som du kan se i kode 22.4 nedenfor, danner grunnlaget for de to spillerne. Klassen har egenskapene `x` og `y`, som representerer punktet øverst til venstre på spiller-rektangelet, og `bredde` og `hoyde`, som bestemmer størrelsen til spilleren. Spillerne får også en fart, en tast som styrer spilleren oppover, en tast som styrer spilleren nedover, og en farge. Tastene får verdier tilsvarende «keyCode» for tilhørende taster.

Spillerne får også to metoder: `tegn()` og `flytt()`. Metoden `tegn()` tegner opp spilleren, og `flytt()` undersøker om spillerens taster har verdien `true` i arrayen `taster`. Hvis en av spillerens taster blir trykket ned, flyttes spilleren i riktig retning.

Helt nederst i koden oppretter vi to objekter, `spiller1` og `spiller2`, som representerer våre to spillere. Spiller 1 får tastene «W» og «S» som henholdsvis opp og ned, mens spiller 2 bruker piltastene opp og ned.

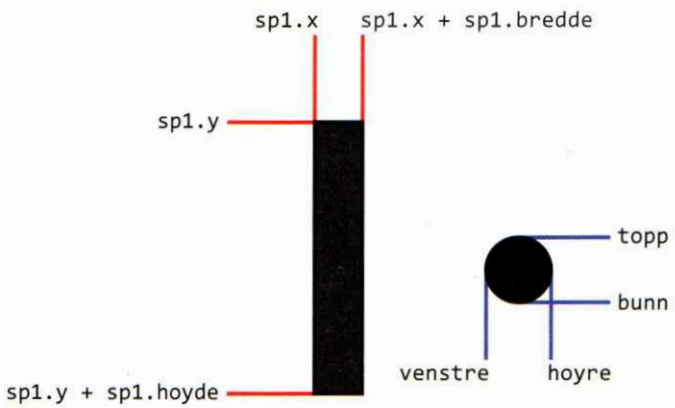
```
// Lager en klasse for spillerne
class Spiller {
  constructor(x, y, bredde, hoyde, fart, opptast, nedtast,
      farge) {
    // Spillerens egenskaper
    this.x = x;
    this.y = y;
    this.bredde = bredde;
    this.hoyde = hoyde;
    this.fart = fart;
    this.opptast = opptast;
    this.nedtast = nedtast;
    this.farge = farge;
  }

  // Metoden tegn() tegner opp spilleren
  tegn() {
    ctx.fillStyle = this.farge;
    ctx.fillRect(this.x, this.y, this.bredde, this.hoyde);
  }

  // Metoden flytt() flytter spilleren
  flytt() {
    this.y += this.fart;

    if (taster[this.opptast]) {
      this.y -= 4;
      this.fart = -4;
    } else if (taster[this.nedtast]) {
      this.y += 4;
      this.fart = 4;
    } else {
      this.fart = 0;
    }
  }
}
```

```
// Lager de to spillerne
var spiller1 = new Spiller(50, 140, 20, 100, 0, 87, 83, "white");
var spiller2 = new Spiller(530, 120, 20, 100, 0, 38, 40,
    "white");
```

Kode 22.4: Klassen `Spiller` bestemmer hvordan spillerne tegnes opp og flyttes.

Det som gjenstår å lage nå, er klassen `Ball`. Du kan se den i kode 22.5 på neste side. Denne klassen bestemmer i utgangspunktet en sirkel som spretter rundt inni `<canvas>`-elementet, men det som er nytt her, er kollisjonene mellom ballen og de to spillerne. Metoden `flytt()` tar to argumenter, `sp1` og `sp2`, som er de to spillerobjektene. Vi må nemlig ha informasjon om spillernes posisjoner for å undersøke om ballen kolliderer med dem. Vi lager variablene `topp`, `bunn`, `venstre` og `hoyre`, som gir oss koordinatene til ballens topp, bunn, venstre og høyre kant, for å forenkle kollisjonstestene vi skal gjennomføre.

For å sjekke om ballen treffer spilleren til venstre, må vi sjekke om ballens venstre kant (`venstre`) ligger til venstre for spillerens høyre kant (`sp1.x + sp1.bredde`), *samtidig* som ballens topp (`topp`) ligger ovenfor spillerens bunn (`sp1.y + sp1.hoyde`), og ballens bunn (`bunn`) ligger nedenfor spillerens topp (`sp1.y`). Denne kollisjonen er illustrert i figur 22.3 nedenfor. Kollisjonen mellom ball og spilleren til høyre følger samme logikk. Når ballen kolliderer med en av spillerne, og hvis spilleren er i bevegelse, overtar ballen y-farten til spilleren den kolliderte med.



Figur 22.3: Illustrasjon av posisjonene involvert i kollisjon mellom ballen og venstre spiller.

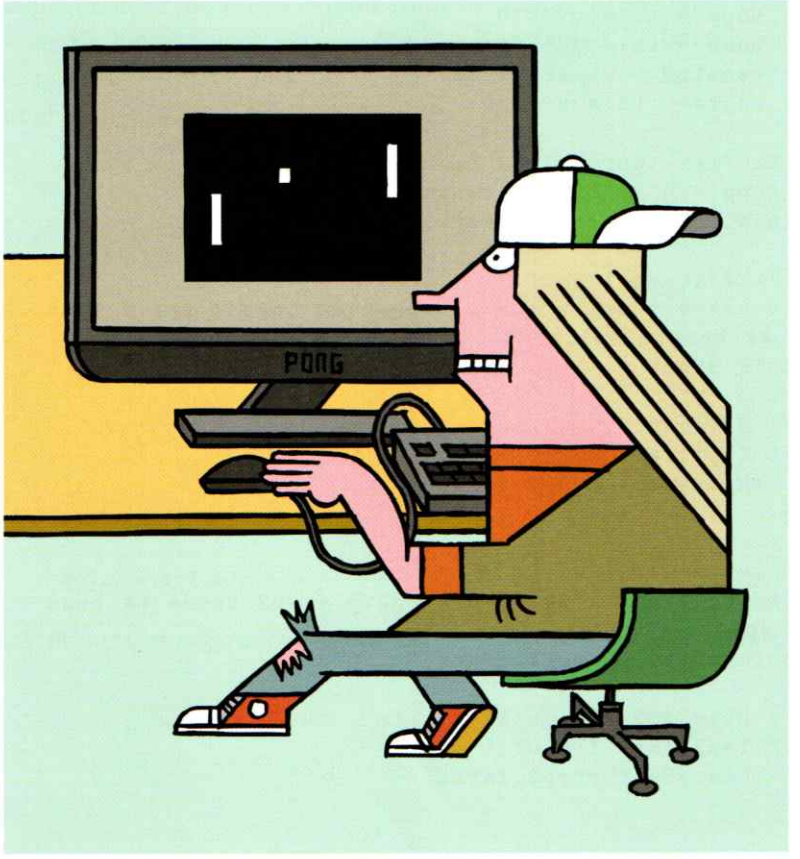
```
// Lager en klasse for ballen
class Ball {
  constructor(x, y, radius, xFart, yFart, farge) {
    this.x = x;
    this.y = y;
    this.radius = radius;
    this.xFart = xFart;
    this.yFart = yFart;
    this.farge = farge;
  }

  // Metoden tegn() tegner ballen
  tegn() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 2 * Math.PI, false);
    ctx.fillStyle = this.farge;
    ctx.fill();
  }

  // Metoden flytt() flytter ballen
  // Argumentene sp1 og sp2 er de to spillerobjektene
  flytt(sp1, sp2) {
    this.x += this.xFart;
    this.y += this.yFart;

    var topp = this.y - 10;
    var bunn = this.y + 10;
    var venstre = this.x - 10;
    var hoyre=  this.x + 10;

    // Treffer topp- eller bunnvegg
    if(topp < 0 || bunn > canvas.height) {
      this.yFart = -this.yFart;
    }
    // Treffer spiller 1
    if(venstre < sp1.x + sp1.bredde && topp < sp1.y + sp1.hoyde
        && bunn > sp1.y) {
      this.xFart = -this.xFart;

      // Hvis sp1 ikke står still
      if (sp1.fart !== 0) {
        this.yFart = sp1.fart;
      }
    }
    // Treffer spiller 2
    if(hoyre > sp2.x && topp < sp2.y + sp2.hoyde && bunn >
        sp2.y) {
      this.xFart = -this.xFart;

      // Hvis sp2 ikke står stille
      if (sp2.fart !== 0) {
        this.yFart = sp2.fart;
      }
    }
    // Forlater "banen"
```

```
    if(this.x < 0 || this.x > canvas.width) {
      this.x = 300;
      this.y = 200;
    }
  }
}

// Lager ballen
var ball = new Ball(300, 200, 10, 5, 0, "white");
```

Kode 22.5: Klassen `Ball` bestemmer hvordan ballen tegnes opp og flyttes (og kolliderer).

Feilsøking

I dette prosjektet er det kollisjonene og posisjonen til ballen som er det meste krevende. Det kan derfor være til god hjelp å skrive til konsollen hver gang ballen forlater banen, treffer toppen eller bunnen av banen eller en av spillerne. Det er også mye enklere å løse utvidelsene på neste side dersom du bruker konsollen aktivt.



Forslag til utvidelser

Som ekstra utfordringer kan du prøve å forbedre pong-spillet med de syv forslagene nedenfor.

Hindre spillerne i å forlate spillet

I versjonen vi har laget, kan spillerne forlate spillet oppover og nedover. Utvid programmet slik at spillerne stopper når de når toppen eller bunnen.

Registrer poeng

Registrer poeng til en spiller hver gang ballen forlater rammen, og vis poengsummene i eller utenfor `<canvas>`-elementet.

Avslutt spillet

Avslutt spillet når en av spillerne har fått et visst antall poeng.

Knapper

Legg til knapper utenfor `<canvas>`-elementet som lar spillerne starte, pause og restarte spillet.

Legg til lyder

Legg til lyder når en spiller treffer ballen, når en spiller ikke klarer å stoppe ballen, og når en spiller vinner spillet.

Ekstraelementer

Legg til ulike ekstraelementer i spillet. For eksempel at ballen går litt raskere etter lange spill, eller små gjenstander man kan treffe for å øke farten til ballen.

Alt du kan tenke deg

Husk at du kan gjøre hva du vil! Du kan for eksempel gjøre om alle størrelser, farger og hastigheter.