Tetris

Skrevet av: Kine Gjerstad Eide

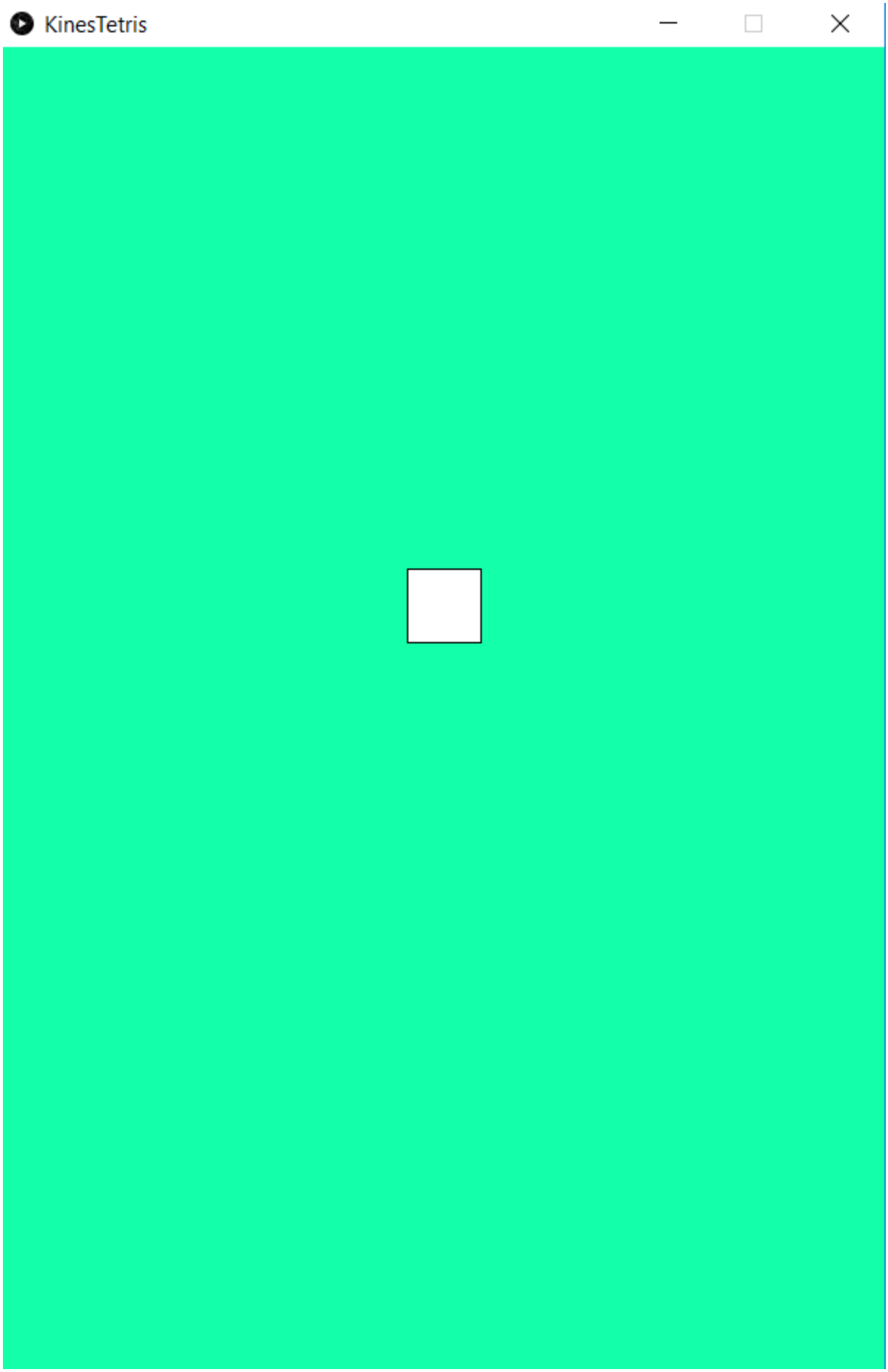
Kurs: Processing

Introduksjon

Lag starten på ditt eget tetris spill!

Det du skal gjøre i denne oppgava er først å sette opp bakgrunnen til spillet og så rett og slett å få firkanter til å falle over skjermen.

Slik vil det se ut når du er ferdig med oppgava:


KinesTetris

Steg 1: Lag dine første to metoder

For hvert steg kommer det forklaring med eksempel og på slutten av steget kommer et bilde av hele koden. Bruk bildet av koden dersom du trenger det, men forsøk å skriv koden på egenhånd ved hjelp av forklaringen først.

Første gjøremål

- [ ] La oss starte med å sette opp et vanlig vindu, dersom du ikke har åpna Processing, så må du gjøre det nå.

- [ ] Lagre programmet ditt, dette gjør du ved å velge `File` og deretter `Save As`.

Det anbefales at du gir koden din et navn som har noe med spillet å gjøre, slik at det er lett å finne igjen. Det er også lurt å lagre koden på et sted som er lett å huske.

Processing er en kodeeditor, det betyr at du kan skrive kode i den. Den første koden du skal skrive er en kodesnutt med to metoder. En metode ser ut slik som dette:

```
void setup(){

}
```

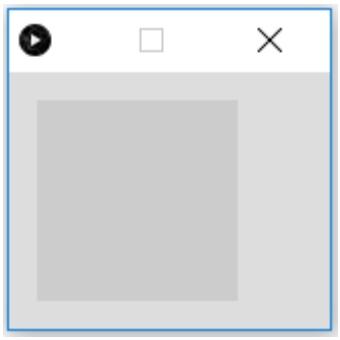
Denne metoden heter `setup`. Alt som skrives inni metoden med navn `setup` skjer én gang når programmet starter, mens det som står inni `draw` skjer på nytt og på nytt helt til programmet avsluttes.

Prøv selv

- [ ] Forsøk å skrive begge metodene selv. De to metodene skal være helt like bortsett fra navnet.

- [ ] Nå må du kjøre koden for å sjekke at metodene fungerer. For å kjøre koden trykker du på **play**-knappen opp til venstre i Proccesing programmet.

Dersom all koden din er riktig skal du få opp et lite vindu som ser slik ut:



Her er hele koden slik den skal se ut nå:

```
void setup(){

}

void draw(){

}
```

- [ ] Dobbelsjekk at du har skrevet alt riktig før du går videre.

Steg 2: Vindustørrelse

Nå skal du bestemme størrelsen på vinduet, det gjør du ved å bruke en metode som noen allerede har skrevet ferdig. Når vi bruker en metode, så skriver vi navnet på metoden, deretter må vi ha parenteser, og til sist legger vi til semikolon. Når man bruker en metode som allerede er ferdigskrevet, så sier man at man *kaller på en metode*.

Her er et eksempel som kaller på metoden med navn `size` :

```
size();
```

Mellom parentesene må man ofte sette inn informasjon, dette kalles parameter. Parameterne som må oppgis når man setter størrelsen på et vindu er rett og slett hvor stort vinduet skal være. Hvis metoden trenger flere parametre, bruker man komma `,` for å skille de:

```
size(100, 200);
```

I eksempelet over er `100` og `200` parametre.

Parametrene som bestemmer størrelsen på vinduet er oppgitt i piksler, så det kommer litt ann på skjermen din hvor mange piksler du ønsker at vinduet skal være. Prøv for eksempel med 600 og 900, og så kan du endre ett og ett tall helt til du får et vindu som du synes har riktig størrelse.

Slik kaller du på metoden som bestemmer størrelsen på vinduet:

```
size(600, 900);
```

Denne kodelinja må skrives inni `setup`, det betyr mellom de to krøllparentesene `{` og `}` som står etter `setup()`.

Hvor mange parametre?

Når man kaller på en metode er det ikke alltid man vet nøyaktig hvilke parametre som skal legges ved. Da kan man enten se i manualen (https://processing.org/reference/), søke etter svaret på internett, eller gjøre et metodekall uten parametre. Gjør du sistnevnte, vil du få opp en feilmelding nederst på skjermen. Feilmeldingen forteller hvor mange og hvilken type parametre som skal skrives mellom parentesene `(` og `)`. Hvert parameter skilles av kommategn.

Prøv selv

- [ ] Lag et vindu som er kvadratisk (like høyt som bredt).
- [ ] Lag et vindu som når helt fra høyre til venstre side av skjermen din.
- [ ] Lag et vindu som er ca. like stort som et russekort.
- [ ] Lag et vindu som du synes er passelig for å spille tetris i.

Steg 3: Bakgrunnsfarge

På samme måte som `size` så er det også en ferdigskrevet metode som kan bestemme bakgrunnsfarge. Denne heter `background` og trenger enten ett eller tre parametre for å fungere. Alle parameterne må være heltall, eller `int` som det heter når man programerer. Tallene som brukes i `background` må være mellom 0 og 255. Tallene bestemmer hvor sterk lyspærene inni pc-skjermen skal lyse.

Hvorfor heter det int?

Heltall er tall som skrives uten komma, slik som 3, 110 og 77. Heltall heter på engelsk *integer*. `int` er altså en forkortelse for *integer* som betyr heltall.

Dersom man bare har en parameter, så får man hvit farge ved å skrive 255, svart ved å skrive 0. Hva tror du det blir for 128? Hva med 200? Her er et eksempel:

```
background(70);
```

Når du bruker tre tall, kaller vi det RGB-farger. RGB står for rød, grønn og blå. Her er et eksempel:

```
background(20, 255, 170);
```

Det betyr at det første tallet `20` bestemmer styrken på det røde lyset, det andre `255` på det grønne og det siste `170` på det blå. Ved å endre RGB-tallene kan man blande akkurat den fargen man ønsker.

Husk å avslutt linjen med semikolon. `background` skal slik som `size` også inni `setup` -metoden.

Prøv selv

- [ ] Test med å først bare ha ett tall inni parentesen og endre på tallet, hva får du? Endre tallet minst 4 ganger og sjekk hvilken farge du får.
- [ ] Få bakgrunnsfargen til å bli svart.
- [ ] Få bakgrunnsfargen til å bli helt lysegrå.
- [ ] Få bakgrunnsfargen til å bli hvit.

- [ ] Prøv å sett inn tre forskjellige tall, og så endrer du på ett og ett av disse, hva skjer?
- [ ] Hvilken farge får du dersom alle parametrene til `background` er 0?
- [ ] Hvilken farge får du dersom alle parametrene til `background` er 255?
- [ ] Få bakgrunnsfargen til å bli rød.
- [ ] Få bakgrunnsfargen til å bli gul.
- [ ] Prøv tilfeldige tall og se hvilken farge du får.
- [ ] Finn en bakgrunnsfarge du liker og gå til neste steg.

Her er et bilde av hvordan koden din skal se ut nå. Husk at du sikkert har funnet litt andre tall enn vi har.

```
int posisjonY;

void setup(){
  size(600, 900);
  background(20, 255, 170);
}

void draw(){

}
```

Steg 4: Lag en firkant

Nå skal du lage firkanten som senere skal falle over skjermen. For å lage en firkant bruker vi metodekallet `rect`. Dette trenger fire parameter, som bestemmer hvor firkanten skal plasseres og hvor stor den skal være. Metodekallet på `rect` skal du skrive inni `draw`-metoden.

Start for eksempel med disse tallene:

```
rect(275, 10, 50, 50);
```

Prøv selv

- [ ] Endre de forskjellige tallene slik at du finner ut hva de står for.
- [ ] Endre plassering og størrelse til firkanten slik at den når fra toppen til bunnen av vinduet ditt.
- [ ] Tegn firkanten slik at den står midt i vinduet.
- [ ] La firkanten dekke hele vinduet. Hint: Du kan bruke minus foran tallene.

Steg 5: Gi firkanten farge

Nå skal du sette en farge på firkanten ved å gjøre et metodekall på `fill`. `fill` tar tre parameter, slik som background, men den fargelegger figurer i stedet for hele bakgrunnen. Denne må skrives inni `draw` og den må stå på linja over firkanten.

Prøv selv

- [ ] Forsøk å gjøre firkanten lilla.
- [ ] Gi firkanten en farge du liker.

Her er vår kode så langt, husk at du sikkert har valgt andre parametre.

```
void setup(){
  size(600, 900);
  background(20, 255, 170);
}

void draw(){
  fill(255);
  rect(275, 10, 50, 50);
}
```

Steg 6: Lag en variabel

For å få firkanten til å falle må vi gjøre to ting. Vi må opprette en variabel som endrer seg og så må vi bruke variabelen i firkanten.

Forklaring av variabel

En variabel er noe som kan endre seg. Vi vil jo gjerne at firkanten skal bevege seg nedover skjermen, altså at firkanten skal endre posisjonen. Dette gjør vi ved å putte inn en variabel i stedet for tallet som bestemmer posisjonen til firkanten.

I den virkelige verden finnes det også variabler som man må endre og så har vi andre tall som alltid er de samme. For eksempel i en håndballkamp har man mange tall. Alle spillerne har sitt eget nummer på drakta. Ingen av spillerne skal bytte nummer under kampen, så derfor er nummeret trykt rett på skjorta. Det er også tall som viser hva stillingen i kampen er. Dette er ofte vist på ei svær tavle. Her endres tallene. Det hadde ikke fungert å trykke stillingen på ei skjorte før kampen. Derfor har de laget tavler, slik at stillingen kan oppdateres etterhvert som lagene får poeng. På samme måte, så lager vi variabler i programmering, slik at vi kan oppdatere de etterhvert som noe skjer i programmet. Alle variabler har sitt eget navn, slik som `poeng`, `liv`, `fart` eller lignende. Når de skal oppdateres, skriver vi for eksempel: `liv = 2`.

For å lage en variabel, må vi først sette av plass i minnet til PC-en, slik at den tar vare på et tall. Da trenger ikke PC-en å vite hva tallet er, men den har satt av plassen slik at det er ledig når vi begynner å bruke det.

Når vi skal lage en variabel i Processing, så må vi si hvilken datatype variabelen skal inneholde. Eksempler på datatyper kan være `int` for heltall, `float` for desimaltall, eller `String` for tekst.

Vi starter med å sette av plass i minnet, det heter å deklarere, eller opprette variabelen. Typen skal være `int`. Vi må gi et navn til variabelen, vi har valgt `posisjonY`, men du kan velge hvilket navn du ønsker. Kodelinja ser slik ut og skal stå over `setup`-metoden:

```
int posisjonY;
```

Så skal vi bestemme hva `posisjonY` skal være når vi starter programmet, det skriver vi inni `setup`-metoden. Kodelinja skal se slik ut (husk at dersom du må bruke samme navn som på forrige kodelinje):

```
posisjonY = 20;
```

Så putter vi variabelen inn i firkanten vår. Det er argument nummer to som må byttes ut, og kodelinja hvor vi skriver firkanten ser slik ut:

```
rect(275, posisjonY, 50, 50);
```

Prøv selv og sjekk at dette fungerer før du fortsetter med steg 7 under.

Prøv selv

- [ ] Bytt ut `20` med for eksempen `200` i `setup`-metoden der du skriver hva `posisjonY` skal være. Hva skjer?

- [ ] Sett `posisjonY` til tallet som får firkanten til å bli plassert helt på bunn av vinduet.

- [ ] Plasser firkanten så langt opp at du bare ser bunnen av firkanten. Hint: Du kan bruke negative tall.

Her er koden så langt.

```
int posisjonY;

void setup(){
  size(600, 900);
  background(20, 255, 170);
  posisjonY = 20;
}

void draw(){
  fill(255);
  rect(275, posisjonY, 50, 50);
}
```

Steg 7: Beveg firkanten

Nå må vi få firkanten til å bevege seg. Som vi skrev helt til å begynne med, så fungerer `draw` metoden slik at den repeteres. All koden inni `draw` blir lest gjennom fra toppen og til bunn. Når programmet er kommet til bunn av `draw`, da bare hopper det opp til starten og leser gjennom koden en gang til. Dette skjer helt til vi avslutter programmet.

For å få firkanten til å endre seg, så må vi oppdatere `posisjonY`. Det gjør vi ved å skrive dette inni `draw`-metoden:

```
posisjonY = posisjonY + 1;
```

For hver gang programmet leser denne linja, så endrer den posisjon til å bli det samme som det den allerede er, pluss en. Det betyr at dersom posisjonen er 20, så vil den bli til 20 + 1 = 21 neste gang, og så 21 + 1 = 22 gangen etter og slik fortsetter det. Når man bruker ett `=` tegn i koding, så betyr det at man gir noe en verdi, ikke at tallene på begge sider av `=` tenget er det samme.

Kjør programmet og sjekk at firkanten faller.

Prøv selv

- [ ] Hvorfor tror du det blir tegna en stripe over skjermen?
- [ ] Hva skjer om du setter `+ 10` i stedet for `+ 1` når du endrer `posisjonY`?
- [ ] La endringen i `posisjonY` være større en høyden på firkanten. Hva skjer da?

- [ ] Finn en fart firkanten kan falle i som du synes er passelig.

Her er hele koden så langt:

```
int posisjonY;

void setup(){
  size(600, 900);
  background(20, 255, 170);
  posisjonY = 20;
}

void draw(){
  fill(255);
  rect(275, posisjonY, 50, 50);
  posisjonY = posisjonY + 1;
}
```

Steg 8: Fjerne stripa som firkanten lager

Grunnen til at det blir tegna en stripe over skjermen er fordi bakgrunnsfargen i vinduet bare blir tegna en gang i `setup`-metoden. For å få det til å se ut som firkanten faller må metodekallet `background` bli flytta inn i `draw` metoden. Det er vanlig å kalle på `background` helt først i `draw` metoden.

Prøv selv

- [ ] Flytt kallet på `background` frem og tilbake fra `draw` til `setup` noen ganger og vær sikker på at du forstår hva som skjer.

- [ ] Kall på `background` øverst i `draw`-metoden.

Her er hele koden:

```
int posisjonY;

void setup(){
  size(600, 900);
  posisjonY = 20;
}

void draw(){
  background(20, 255, 170);
  fill(255);
  rect(275, posisjonY, 50, 50);
  posisjonY = posisjonY + 1;
}
```

Lisens: CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0/deed)