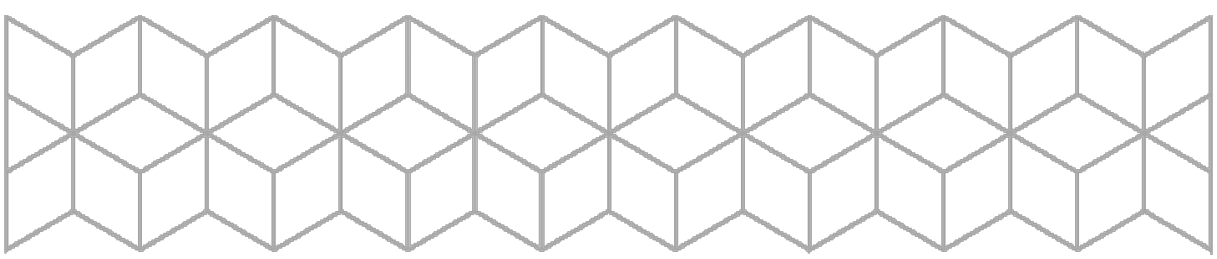
Høgskolen i Østfold

SENSORVEILEDNING

Emnekode:	ITF19019
Emnenavn:	**Innføring i Programmering**
Eksamensform:	**Skriftlig, 4 timer**
Dato:	29. november 2019
Faglærer(e):	Tore Marius Akerbæk Joakim Karlsen
Eventuelt:	



Om evalueringen

Kurset har hatt fokus på å etablere forståelse for algoritmisk tankegang, og å lære studentene å bryte ned utfordringer og problemer i mindre delproblemer (så iterativt som mulig). Deretter kan løsningene appliseres på delproblemene for å arbeide seg oppover i problemhierarkiet.

Evalueringen skal ta med forståelse og evnen til å bryte ned problemer som en del av evalueringen.

Emnebeskrivelse for kurset finnes på høgskolens nettsider, se https://www.hiof.no/studier/emner/it/2019/host/itf19019.html

Om innholdet

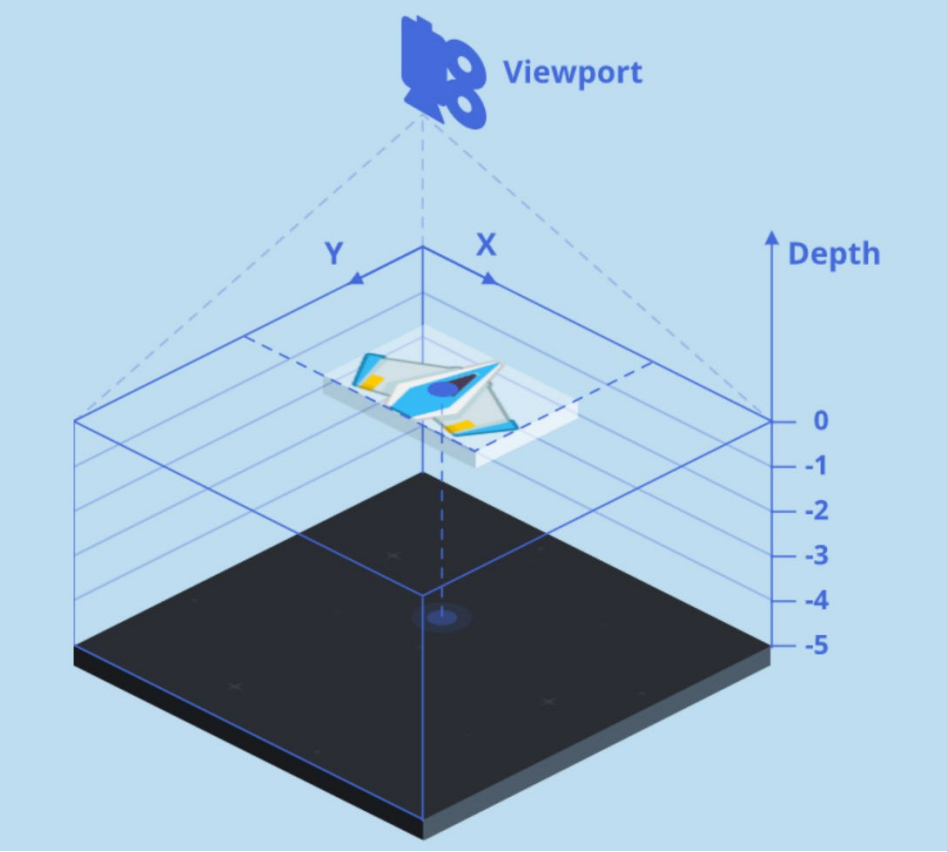
I kurset er det benyttet et verktøy kalt CT.js, som er en spilleditor bygget på rammeverket Pixi.js. CT.js er et skall over JavaScript (ES6), og eksamensoppgaver og -besvarelser er tillatt gitt med kode fra CT.js, siden dette ivaretar grunnleggende programmeringsprinsipper.

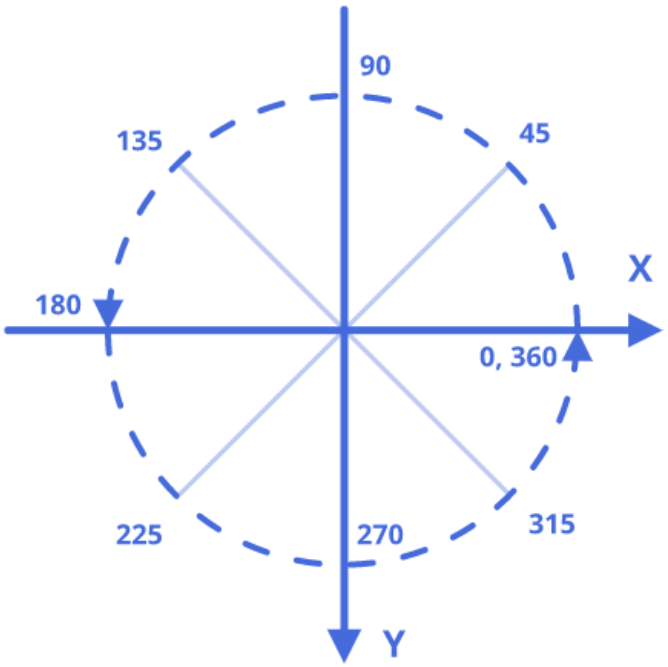
For eksterne sensorer vil det være til hjelp som en del av forberedelsen å laste ned CT.js (https://ctjs.rocks/download/) og gjennomføre de to tutorialene fra dokumentasjonen (SpaceShooter: https://docs.ctjs.rocks/tut-making-shooter.html og Platformer: https://docs.ctjs.rocks/tut-making-platformer.html)

Videre følger et løsningsforslag til oppgavene i den skriftlige eksamen.

Løsningsforslag

Til hjelp: i CT.js og oppgavene i eksamen opererer vi med disse forutsetningene for størrelse/aksediagram og retning.





Oppgave 1

Denne oppgaven går på kodelesing og -forståelse. Du blir presentert for noen kodesnutter (og dersom nødvendig, kontekst) som du skal redegjøre for hva gjør eller resulterer i.

Oppgave 1.1

På bildet under vises en liten kodesnutt som inneholder fire grunnleggende prinsipper innenfor programmering. Navngi disse fire prinsippene.

On Create | On Step

```
1 for(var krystall of ct.types.list.GreenCrystal) {
2     sjekkKrystaller(krystall);
3 }
4
5
```

1 2 3 4

Løsningsforslag:
1: variabel
2: løkke / liste
3: funksjon
4: parameter (variabel)

Oppgave 1.2

```
1 if(ct.place.occupied(this, this.x, this.y + 1, 'Solid')) {
2     if(ct.actions.Jump.down) {
3         this.vspd = this.jumpSpeed;
4     } else {
5         this.vspd = 0;
6     }
7 } else {
8     this.vspd += this.gravity * ct.delta;
9 }
```

Forklar, linje for linje, hva koden på bildet over gjør. Forutsetningene for koden er at det er laget en Action i CT.js kalt Jump som er knyttet til tastaturknappen Space, og at this.jumpSpeed er deklarert på typen ved opprettelse (on Create).

Løsningsforslag

Linje 1 sjekker om posisjonen under denne typen (this.y + 1) er okkupert av en annen type med kollisjonstype «Solid».
Linje 2 sjekker om Space-tasten er trykket ned.
Linje 3 setter denne typens vertikale hastighet til å være det samme som verdien av variabelen jumpSpeed
Linje 4 avslutter testen om Space-tasten er trykket ned, og lager en forutsetning for andre tilfeller («else»; ellers så…)
Linje 5 setter denne typens vertikale hastighet til 0 (siden Space-tasten ikke er trykket ned)
Linje 6 avslutter denne else-forutsetningen
Linje 7 avslutter testen om «Solid»-okkupasjonen fra linje 1, og lager en forutsetning dersom plassen ikke er opptatt av en «Solid»-type
Linje 8 setter denne typens vertikale hastighet til å være verdien av gravitasjon * delta, som betyr hvor fort typen skal ramle mot bakken (this.gravity) med justering for manglende frames i spillmotoren (ct.delta).

Oppgave 1.3

```
1 if(ct.place.meet(this, this.x, this.y, 'Robot')) {
2     ct.room.crystals++;
3     this.kill = true;
4 }
```

Forklar, linje for linje, hva koden på bildet over gjør. Beskriv også hvilke forutsetninger som ligger til grunn for at koden skal fungere (avhengigheter til andre typer, deklarerte variabler o.l.).

Løsningsforslag

Linje 1 gjør en sjekk om denne typen møter/treffer en type kalt «Robot» på en x,y-posisjon
Linje 2 legger til 1 i verdien som er lagret i variabelen ct.room.crystals
Linje 3 «dreper» (fjerner) denne typen
Linje 4 avslutter sjekken i linje 1

Forutsetninger: Det må finnes en type Robot, og en deklarert variabel Crystals som inneholder tall.

Oppgave 1.4

```
1 for(var i = 0 i < 5; i+) {
2     rcx = ct.radnom.range(0, ct.viewWidth);
3     rcy = ct.random.range(0, ct.viewHeight);
4 
5     if(ct.place.free(GreenCrystal', rcx, rcy, 'Solid')) {
6         ct.types.copy('GreenCrystal', rcx, rcy)
7 
8 }
```

I koden på bildet over er det seks feil som gjør at koden ikke vil fungere. Finn og beskriv feilene.

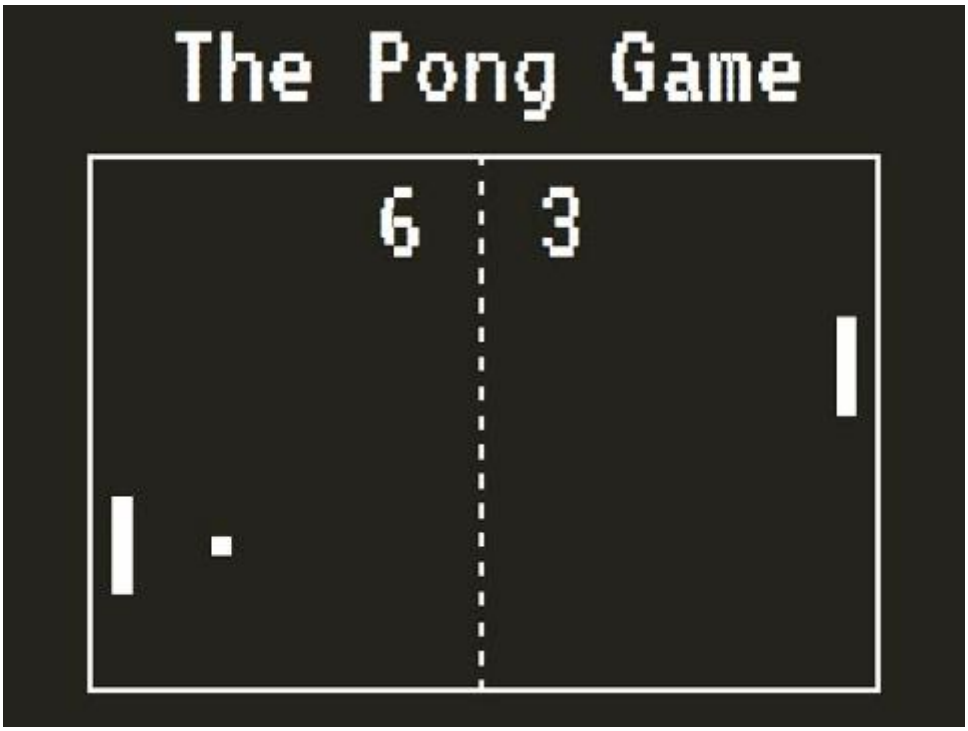
Løsningsforslag

Linje 1: Det mangler et semikolon etter i = 0. Det mangler et plusstegn etter i+
Linje 2: random er feilstavet
Linje 5: Det mangler en startende fnutt foran GreenCrystal
Linje 6: Det mangler et semikolon etter ct.types.copy()-funksjonen.
Linje 7: Det mangler en avsluttende klammeparentes for å avslutte testen på linje 5 (eller løkken på linje 1, som koden ville lest)

Oppgave 2

I oppgave to blir du presentert for noen utfordringer som skal løses ved hjelp av programmering. I svarene trenger du ikke skrive programkode, men planlegge og forklare hvordan utfordringene kan løses ved å bryte den ned i delproblemer med egne løsninger.

I bildet under ser vi spillklassikeren Pong. Ikke ulikt bordtennis (ping pong) består spillet av to spillere mot hverandre med hver sin padel/racket, og en ball. Ballen spretter mot toppen/bunnen av spillbrettet, og det er om å gjøre å slå ballen mot den andre spilleren. Hvis man ikke treffer ballen med padelen/racketen sin, forsvinner ballen ut av spillbrettet og spilleren som slo ballen sist får et poeng.



2.1 Ta for deg utfordringen med spillere og deres bevegelser. Beskriv utfordringer/funksjoner knyttet til disse, og bryt dem ned for å finne potensielle løsninger.

Løsningsforslag

Hver spiller skal kun kunne **bevege seg vertikalt**, hvilket betyr at vi kun trenger å tenke på kontroller og bevegelse **langs Y-aksen** i koordinatsystemet. Spillerne skal **ikke kunne bevege seg på utsiden av spillbrettet**, altså må spilleren stoppe når den når en av langsidene. Vi må ha **input-funksjonalitet** for fire bevegelser; spiller 1 opp, spiller 1 ned, spiller 2 opp og spiller 2 ned.

2.2 Ta for deg utfordringen med ballen og dens bevegelser. Beskriv utfordringer/funksjoner knyttet til dette, og bryt dem ned for å finne potensielle løsninger (Ballens vinkler skal diskuteres, men det forventes ikke at den matematiske utregningen redegjøres for).

Løsningsforslag (Her finnes det flere ulike løsninger. Fokuser på evnen til å detektere utfordringer og tanker rundt mulige løsninger).

Ballen må **starte et sted**. Den kan serves av en spiller (avansert) eller droppes vilkårlig ned på spillbrettet (enklere). Vi tar utgangspunkt i at ballen starter i midten av spillbrettet. Den må **bevege seg i en retning** fra starten av spillet. Denne kan vi forhåndsbestemme (samme hver gang), eller randomisere (uventet utfall). Ballen bør ikke går rett på en spiller (0 eller 180 grader ihht gradeskiven for CT.js), så her bør vi **begrense antall vinkler ballen kan starte i**. Kanskje det vil være greit å holde seg innenfor 45-135 grader oppover og 225-315 grader nedover. Startvinkelen vil ha noe å si for hvor ballen beveger seg videre. Hvis den eksempelvis starter med en vinkel på 45 grader (oppover mot høyre), vil den måtte fortsette nedover mot høyre når den **treffer toppen av spillbrettet**. På samme måte må den **skifte retning når den treffer en spiller**. Treffer den spiller to til høyre, skal retningen nå endres til å sprette mot venstre side av spillbrettet (og motsatt når den treffer spiller en). Hvis en **spiller ikke treffer ballen, skal ballen forsvinne** ut kortsiden av spillbrettet, og en **ny ball** skal startes på spillbrettet.

2.3 Ta for deg poengsystemet. Beskriv regler og oppførsel spillet må håndtere for å få et fungerende poengsystem.

Løsningsforslag:

Begge spillere **starter med 0 poeng**. Et **poeng gis** en spiller når motspilleren ikke klarer å treffe ballen med sin «padel», og **ballen forsvinner ut av spillbrettet**. Det vil si at dersom ballen går ut på høyre side av spillbrettet (hvor spiller to befinner seg), skal spiller en få ett poeng. Det må finnes **to poengtellere, en for hver spiller**. Denne **må lagres** for hvert spill, og for **hver gang ballen går utenfor spillbrettet**, må vi **sjekke** hvilken spiller som bommet på ballen, og gi **pluss ett poeng på telleren til motspilleren**.

2.4 Hvis spillet skulle lages, hvilke typer (i CT.js) / objekter måtte vi ha tilgjengelig for å kunne lage spillet? PS: Husk at utskriften av poengene også regnes som typer.

Løsningsforslag:

Vi måtte ha typer for

1. Ball
2. Spiller 1
3. Spiller 2
4. Spiller 1 poeng
5. Spiller 2 poeng

Oppgave 3

I oppgave tre skal du selv kode noen løsninger for noen gitte utfordringer. Du kan her presentere løsningene som kode i CT.js eller ren JavaScript. Dersom du er usikker på syntax, bruk pseudokode og kommenter underveis i koden for å tydeliggjøre hva du ønsker å oppnå med koden.

Oppgavens grunnlag

Vi skal lage et labyrintspill. Nivå 1 er et spillbrett som er 600 x 600 pixler stort, og heter «Level1». Vi har en type kalt «Blokk» som illustrerer steiner på 30 x 30 pixler, som en type kalt «Spiller» ikke skal kunne gå gjennom (collision type «Solid»). «Spiller» starter nede i venstre hjørne av spillbrettet, og en type kalt «Utgang» (døren til neste nivå) er plassert øverst til høyre.

Oppgave 3.1

Skriv et forslag til kode som sørger for at typen «Spiller» ikke beveger seg på utsiden av spillbrettet.

Løsningsforslag

På «Spillers» on step-kode:

```
//Stopp spiller langs x-aksen
If(this.x < 0)  {
    this.x = 0;
}
If(this.x > ct.viewWidth) {
    this.x = ct.viewWidth;
}

//Stopp spiller langs y-aksen
If(this.y < 0) {
    this.y = 0;
}
If(this.y > ct.viewHeight) {
    this.y = ct.viewHeight;
}
```

Oppgave 3.2

Skriv et forslag til kode som plasserer 20 kopier av typen «Blokk» vilkårlig rundt på spillbrettet.

Løsningsforslag:

Som et minimum bør det ligge noe ala dette

```
//For-løkke for å gjøre handlingen 20 ganger.
For(b = 0; b < 20; b++) {
    //Bruker random for å lage vilkårlige X- og Y-posisjoner
    var pos_X = ct.random.range(0, 600);
    var pos_Y = ct.random.range(0, 600);
    //Bruker ct.types.copy til å lage en kopi av typen 'Blokk' og plassere den på
den genererte posisjonen fra steget over.
    ct.types.copy('Blokk', pos_X, pos_Y);
}
```

Ideelt skal ingen av blokkene ligge oppå hverandre, ergo bør det testes for kollisjon på den genererte posisjonen før man oppretter en ny blokk. Dersom den random genererte plassen er opptatt av annen blokk, må det gjennomføres en ny sjekk. Dette krever en rekursiv funksjon.

I tillegg lar ikke CT.js oss sjekke kollisjon på elementer som ikke eksisterer. Dette betyr at vi må lage de 20 blokkene først, og deretter gå gjennom listen med de eksisterende blokkene, og for hver blokk generere random posisjon, sjekke om denne er ledig, og deretter plassere blokken – alternativt lage ny random posisjon og sjekke denne (rekursivt, se over).

Med disse forutsetningene blir koden omtrent slik:

```
//Generer 20 blokker utenfor spillbrettet
for(b = 0; b < 20; b++) {
    ct.types.copy('Blokk', -1000, -1000);
}

//Lag en funksjon som tar imot en type for å sjekke (og flytte til) posisjon
function sjekkOgFlytt(enBlokk) {
    random_X = ct.random.range(0, 600);
    random_Y = ct.random.range(0, 600);

    if(ct.place.free(enBlokk, random_X, random_Y, 'Solid') {
        enBlokk.x = random_X;
        enBlokk.y = random_Y;
    } else {
        sjekkOgFlytt(enBlokk);
    }
}

//Deretter, løp gjennom blokkene vi lagde utenfor spillbrettet, og kjør funksjonen
på hver enkelt av dem:
for(var eksisterendeBlokk of ct.types.list.Blokk) {
    sjekkOgFlytt(eksisterendeBlokk);
}
```

Oppgave 3.3

Skriv et forslag til kode som åpner Nivå 2 når «Spiller» kommer til «Utgang». PS: CT-funksjonen for å bytte rom er ct.room.switch('Nytt rom').

Løsningsforslag

```
If(ct.place.meet(this, this.x, this.y, 'Spiller')) {
    ct.room.switch('Nivaa_2');
}
```

Oppgave 3.4

Lag to arrayer. Begge arrayene skal inneholde fem verdier innenfor spillbrettets størrelse.

Lag en kode som løper gjennom begge arrayene og plasserer en kopi av «Blokk» for hvert koordinatpar. Et koordinatpar er satt sammen av en verdi fra hver av arrayene.

Løsningsforslag:

```
Var coordsX = [21, 98, 177, 245, 421];
Var coordsY = [574, 498, 355, 276, 120, 75];

For(c = 0; c < 5; c++) {
    ct.types.copy('Blokk', coordsX[c], coordsY[c]);
}
```