Synopsis Synk et Skib

Resume

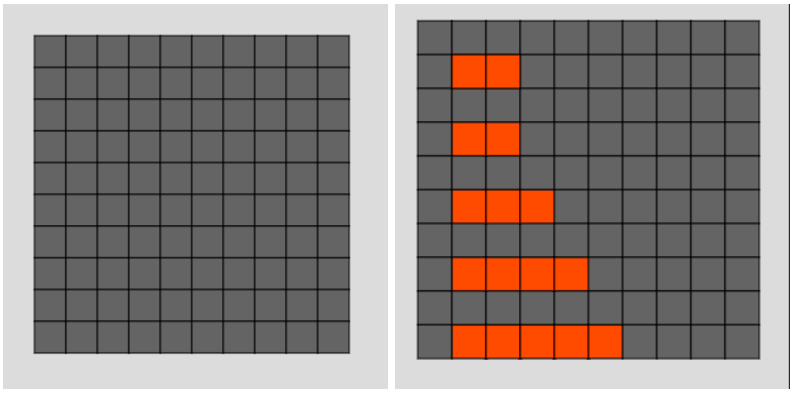
I denne opgave ville vi arbejde med at spille sænke slagskibe og prøve at lave et spil ud fra det. Desværre stødte vi på mange problemer og vores krav og problemformulering blev ikke løst. Dog havde vi mange tanker bag vores nuværende kode. Dette vises både med eksempler fra koden, men også med diagrammer og teori. I denne synopsis kommer jeg også med løsningsforslag til, hvad man kunne have gjort, for at få koden til at virke, samt hvordan vi havde tænkt programmet ville virke og hvilke forbehold vi havde taget med vores nuværende kode, for at få det til at fungere som et stort sammenhængende spil.

Problemformulering

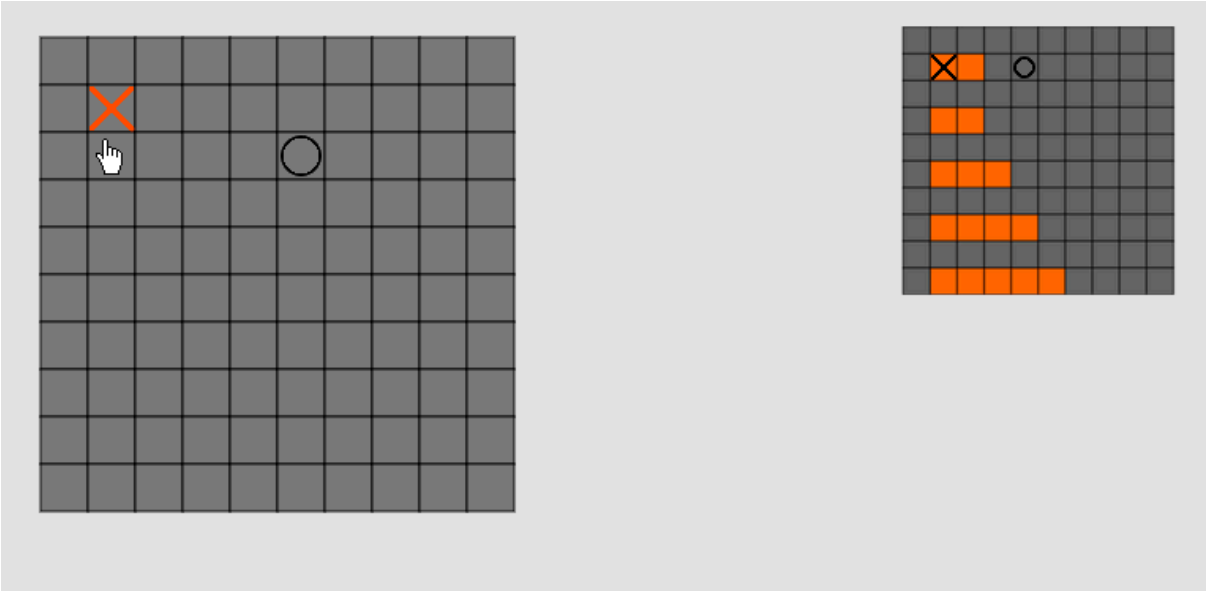
Meningen med programmet, var at lave et spil, som ville minde om sænke slagskibe (battleships) Programmet skal være lavet til en spiller, altså ikke multiplayer.

Funktionsbeskrivelse

Programmet viser en enkelt skærm, hvor der er blevet lavet et gitter af 10*10 kvadrater som vist herunder. Disse kvadrater fungerer tilsammen som spillepladen.



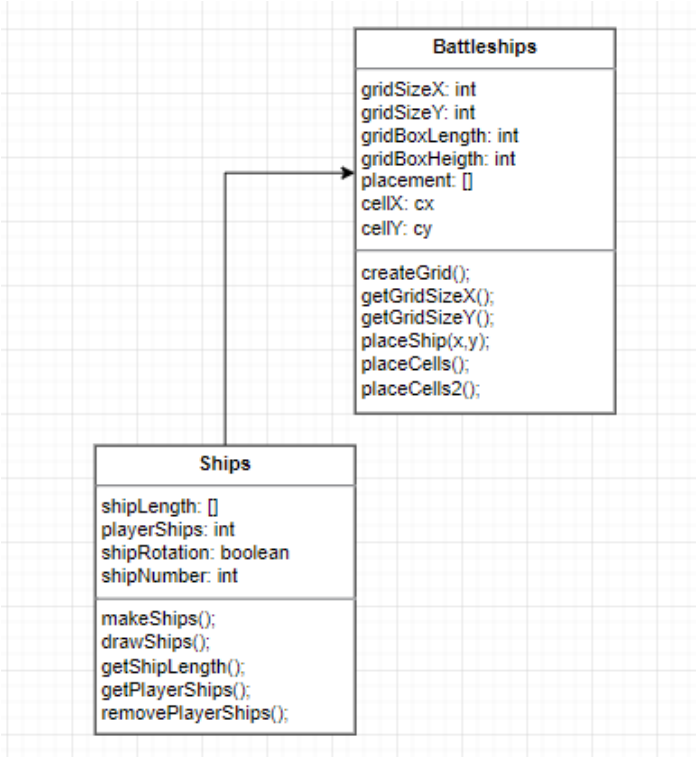
Hvert kvadrat er et punkt, hvor et skib kan placeres. Der er i alt fem skibe med fire forskellige størrelser. Dette vises på det andet billede, hvor de røde kvadrater er skibe. I starten af programmet er der et tomt gitter, som spilleren herefter selv skal fylde ud med skibe. Det er dog kun muligt at placere et enkelt skib med den nuværende kode. Meningen med programmet var, at spilleren skulle trykke fem forskellige steder for at placere de fem skibe på gitteret, som også fungerer som spillerens plade. Skibene skulle ikke kunne placeres oven på hinanden, altså der kan ikke stå to skibe på samme kvadrat. Meningen var også, at der skulle være en knap ved siden af gitteret, som kunne rotere skibene, så de kunne placeres både vandret og lodret. Programmet skulle også have haft en knap, når spilleren havde placeret alle sine skibe. Denne knap skulle sætte spillet i gang, samt placere computerens fem skibe på en anden tilfældig plade med tilfældige placeringer. Det vigtigste her er, at computeren altid vælger tilfældige steder at placere sine skibe, så den ikke ender med at blive forudsigelig. Herefter skulle spilleren trykke på de kvadrater, som de troede der var et skib på.



Herover ses en model af, hvordan det endelige spil skulle se ud, hvis det var færdiglavet. På venstre side af billedet vises en model af modstanderens (computeren) spilleplade. Det røde kryds betyder, at spilleren har ramt et af modstanderens skibe. Den sorte cirkel betyder, at spilleren har ramt et sted, hvor der ikke er placeret et skib. På højre side af billedet vises en model af spillerens egen spilleplade, så man har overblik over, hvor mange skibe man har tilbage, samt hvor modstanderen har skudt. Desuden kan hverken spilleren eller modstanderen skyde det samme sted to gange. Spilleren starter altid med at skyde og efter dette, får henholdsvis modstanderen og spilleren et forsøg hver til at ramme et skib, hvorefter spillet skifter tur. Spillet er færdigt, når enten spilleren eller computeren har ramt alle skibene på modstanderens spilleplade.

Dokumentation af Programmet

I vores projekt valgte vi at bruge objektorienteret programmering, da det ofte kan være en effektiv måde at programmere på, når man arbejder med mange objekter som skal have forskellige værdier, men samme formål. I dette projekt kan det være godt at bruge objektorienteret programmering til at lave skibene.



På ovenstående billede ses et eksempel på et funktionsdiagram til vores program. Den første klasse som bliver lavet i spillet er “Battleships”, som indeholder alle informationer om spillets generelle funktioner og knapper samt spillepladerne og deres placeringer. “Battleships” er en superklasse, da den tager informationer fra andre klasser og bruger dem til at lave spillet. “Ships” er en anden klasse i spillet, som bruges til at danne og tegne skibene i spillet. Denne klasse kaldes for indkapsling, da den bruges til at lave skibe samt finde deres værdier, hvorefter de bliver brugt i “Battleships” klassen.

Den kode, som vi har skrevet i programmet, består af flere forskellige dele, hvor den første er de to indbyggede funktioner, setup() og draw(). Setup() indeholder alle de ting, som skal laves det sekund spillet starter, mens draw() opdaterer dens funktioner hele tiden. I vores program bruges setup() kun til at lave baggrunden og de to klasser. Draw() funktionen bruges også kun til to funktioner, da de metoder som kræver interaktion med musen, skal bruge en anden funktion for at fungere. CreateGrid() og drawShips() er inden under draw(), da begge metoder skal opdateres hvert frame og ikke en enkelt gang, men på samme tid fungerer det uafhængigt af interaktion med musen og musens koordinater. CreateGrid() kunne også være brugt i setup() funktionen, men dette ville skabe problemer senere, hvor spillet begynder og funktionen skal bruges flere gange.

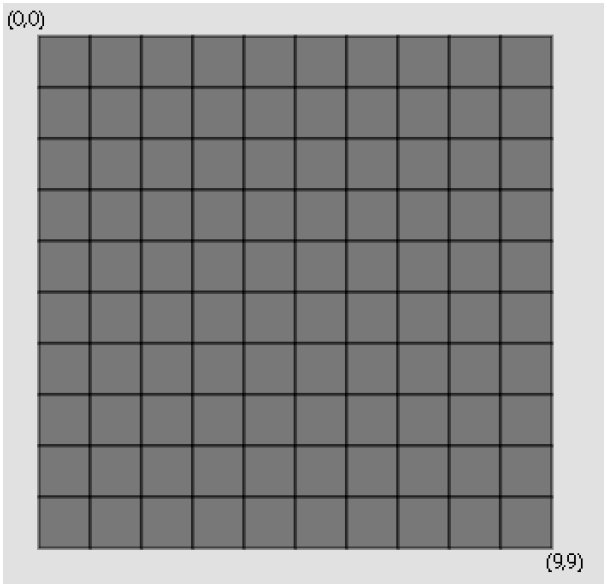
```
function setup() {
  createCanvas(800, 800);
  Battleships = new BattleshipsGame();
  Playerships = new Ships();
}

function draw() {
  background(220);
  Battleships.createGrid();
  Playerships.drawShips();
}
```

Den næste del af koden består af funktionen mouseClicked(), som bliver aktiveret, når spilleren trykker på skærmen. Inde i mouseClicked() har vi sat en if-funktion op, som sikrer at andre metoder kun kører, når der bliver trykket inden for gitteret. Hvis man for eksempel ville lave en knap som roterede skibene, ville denne metode ikke være inde i if-funktionen, da knappen skal være uden for gitteret. If-funktionen er defineret med fire parametre, som hver er et hjørne af gitteret. Dette gør, som nævnt tidligere, at visse metoder kun bruges, hvis musen bliver trykket på inden for gitteret. Her bruges metoderne getGridSizeX() og getGridSizeY(), for at finde gitterets bredde og højde.

```
function mouseClicked() {
  if (mouseX > 30 && mouseX < 30+30*Battleships.getGridSizeX() && mouseY > 30 && mouseY < 30+30*Battleships.getGridSizeY()) {
    Playerships.getShipLength().splice(Playerships.getPlayerShips-1,1);
    print(Playerships.getPlayerShips());
    Playerships.removePlayerShips();
    Battleships.placeShip(mouseX,mouseY);
  }
}
```

I den første metode i mouseClicked() finder koden først det skib, som er blevet placeret, og derefter fjerner den skibet fra listen ved hjælp af splice(), som finder det første skib i listen getPlayerShips og derefter fjerner det, så det næste skib er klar til at blive placeret. Den sidste metode i mouseClicked() er den som placerer skibet det rigtige sted på gitteret. Metoden bruger de indbyggede variabler mouseX og mouseY, som indeholder informationerne om musens position på skærmen.

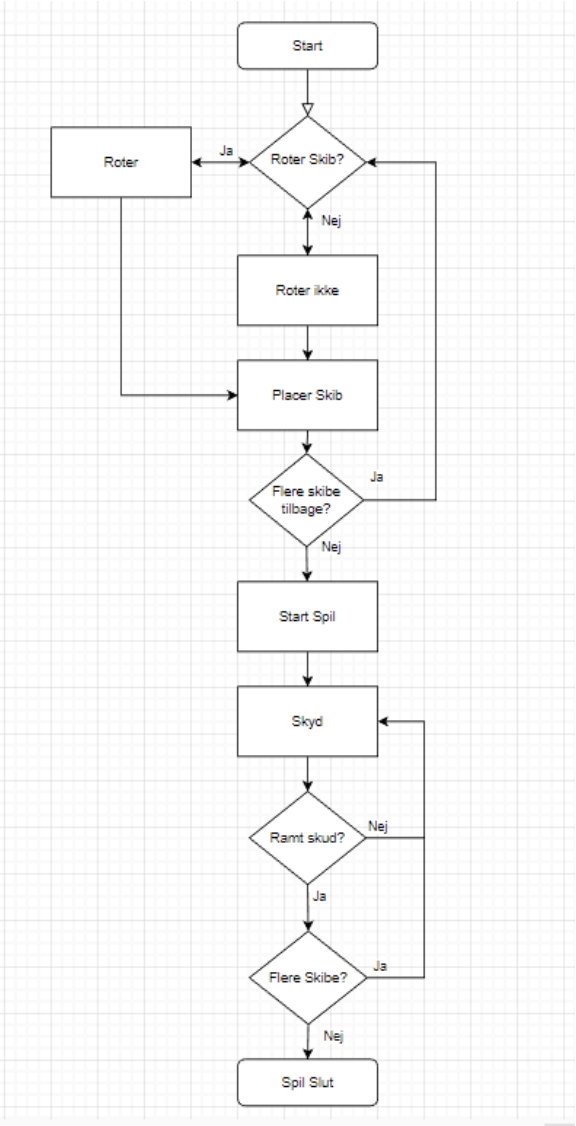
Den første klasse i programmets kode er “Battleships”, som indeholder informationer om hele spillet. Klassen starter med en constructor(), som er en funktion der kan oprette lokale variabler i en klasse. Disse variabler kan senere bruges lokalt inde i klassen “Battleships”. CreateGrid() er en relativt simpel funktion i klassen. Funktionen består af to for-lykker, som kører igennem de to variabler this.gridSizeX og this.gridSizeY, som blev dannet i constructoren. Brugen af variabler her er vigtig, da det gør det nemmere at ændre i koden senere, hvis man for eksempel har lyst til at ændre på gitterets størrelse, samt hvis man vil bruge variablerne andre steder end i den lokale klasse. Ud fra disse to variabler laver createGrid() et gitter på den bestemte størrelse (10*10). En anden vigtig metode i klassen “Battleships” er placeShip(), som nævnt tidligere bruger mouseX og mouseY til at placere skibene. Metoden laver mouseX og mouseY om til nye tal, da skibene ellers både ville blive placeret det forkerte sted men heller ikke centreret i kvadraterne i gitteret. MouseX og mouseY bliver derfor omregnet til koordinater som på et koordinatsystem, som kan ses på billedet under.



Denne omregning gør det nemmere at finde ud af, hvor skibene skal placeres, samt skaber to nye lokale variabler (cellX og cellY). Herefter bliver de to placeringer sat ind i et array ved hjælp af push() funktionen. Arrayet som variablerne bliver tilføjet til hedder this.placement[]. Dette array indeholder alle skibenes placeringer, når de er blevet sat på spillepladerne, hvilket også kan være brugbart, hvis man senere i spillet skulle tjekke om skibet er blevet ramt.

Den anden klasse i koden er “Ships”. Denne klasse indeholder alle informationer om mængden af skibene, samt deres størrelser og rotationer. Metoden makeShips() er ansvarlig for at lave skibene og deres farve. Metoden kører igennem to for-lykker for at finde ud af hvilket skib skal placeres, samt dets størrelse. Dernæst tjekker den om skibet er blevet roteret, og farver derefter den mængde kvadrater, som svarer til skibets størrelse.

DrawShips() er den sidste vigtige funktion i koden, da den står for at skibene bliver placeret det rigtige sted ved hjælp af informationer fra den tidligere funktion placeShip().



Her ses et billede af et flowchart, som skal vise den proces som en spiller skal igennem, for at gennemføre spillet. Det er dog ikke kun spilleren som går gennem dette, men også computeren. Man kan se det som om, at hver gang spilleren har et valg om enten at placere eller rotere et skib, bliver dette valg erstattet af et tilfældigt nummer for computeren. Det første valg en spiller skal tage, er om de vil rotere deres skib, og uafhængigt af dette valg, skal deres skib placeres. Når alle skibe er blevet placeret, vil computeren køre igennem samme proces og spillet starter derefter. Når et skud bliver affyret, vil der blive tjekket, om et skib er blevet ramt, hvis ikke vil spillet skifte tur, og samme proces vil ske indtil alle skibe er blevet ramt. Når enten alle spillerens skibe er blevet ramt, eller når computerens skibe er blevet ramt, vil spillet slutte.

Manglende kode

Lav alle skibe på spillepladen

Som nævnt tidligere, mangler der en del kode i programmet, før spillet er funktionelt. Programmet kan på nuværende tidspunkt kun lave et enkelt skib, hvorefter det prøver at lave det samme skib igen. Hvis man skulle fikse dette, ville det være ved at for det første ændre for-lykken i drawShips(), da den nuværende kode kun kan lave skibe med en størrelse på to kvadrater. Man skulle her ændre den nuværende værdi fra to til det næste tal i arrayet this.shipLength[]. En anden ting man også skulle gøre, for at få skibene til at blive placeret forskellige steder er, at fjerne de to koordinater i arrayet this.placement[], da dette array altid vil tage de to første tal og tegne skibet ud fra dem. På nuværende tidspunkt vil alle skibe blive lavet i den samme størrelse og det samme sted på skærmen.

Computer tilfældige skib placeringer
En anden måde man kunne forbedre koden på ville være, at lave et helt nyt gitter, som ville være computerens spilleplade. Dette ville man kunne gøre ved hjælp af den indbyggede funktion random(). Denne funktion fungerer ved at den får to tal som input, og vil derefter give et tilfældigt tal mellem disse to. Man kunne på denne måde få computeren til at bruge cellX og cellY til at finde et tilfældigt kvadrat, hvor den kunne placere et skib.

Kollision-tjek
Den sidste og en ret væsentlig ting, er at lave et kollisionssystem, som ville have to vigtige funktioner. For det første ville den kunne tjekke, om der allerede er et skib på, det sted hvor man prøver at placere et skib, så der ikke kommer til at være to skibe på samme kvadrat. Dette kunne man gøre ved at bruge det array som indeholder alle skibenes placeringer, altså this.placement[]. Dette array ville også kunne bruges til det næste kollisions tjek. Dette ville være her hvor man kan skyde hinandens skibe, og man bruger her de samme koordinater til at udregne, om skuddet har ramt et skib eller ej. Ud fra dette ville man kunne få en boolean værdi som enten er true eller false, og gitteret ville her enten få et kryds, fordi man har ramt et skib, eller en cirkel fordi man har ramt forbi.

Test af programmet
Som vist og nævnt tidligere, blev programmet ikke hvad vi havde regnet med, men selvom mangel på visuelle beviser er det stadig muligt at komme med en lille test af det vi fik opnået. Både link til programmet og en kort video demonstration findes i bilag.

Konklusion
I dette projekt fik vi ikke opnået vores mål om at lave sænke slagskibe, men vi fik lagt et grundlag for et spil som ville kunne spilles med mere arbejde på koden. Vi kom med flere løsningsforslag til, hvordan det ville være muligt at færdiggøre spillet. Og hvordan det ville være muligt at løse nogle af de problemer, som vi stødte på gennem arbejdsprocessen. Dog viste vi, at det ville være muligt at arbejde videre med spillet, da vi har opsat flere variabler og arrays, som kan bruges på senere tidspunkter og ville gøre en eventuelt senere arbejdsproces nemmere og mere overskueligt.

Bilag

Program:
https://editor.p5js.org/alexanderhougaard123/full/24Ftnret0

Program med kode:
https://editor.p5js.org/alexanderhougaard123/sketches/24Ftnret0

Video af Program:
https://youtu.be/AfuObk_qawU