Indhold

1 Sekventiel/procedural -programmering

Læs kapitel 3.1 i Systimebogen.
Denne opgave handler om sekventiel og procedural programmering. Sekventiel programmering, betyder at instruktionernes rækkefølge ikke er ligegyldig. Procedural programmering betyder at vi kan opdele vores program i forskellige procedurer eller funktioner og bare kalde proceduren når vi har brug for den. Det ville være smart med en procedure som kan beregne en vares pris med moms.

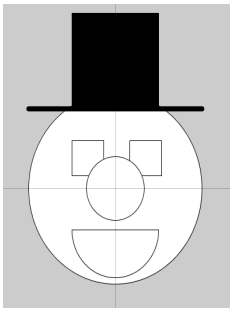
Det vigtige i denne opgave er, at du skal lære at bruge dokumentationen til processing/Java.

Herudover vil du blive introduceret til rutediagrammer. Et værktøj som skal hjælpe dig med at strukturere dine programmer.

Der er altså hele tre ting som kræver din opmærksomhed.

1.1 Opgave

Du skal lave en kopi af min tegnig: opg1-hoejhat.pdf. Det primære fokus i denne opgave er, at bruge dokumentationen i processing. Der findes i processing en lang række forud definerede procedurer/funktioner. Vi kan se at det er en funktion fordi er altid er () bagefter. For eksempel size(); Size er en funktion som kræver to parameter, brede og højde. size(400,600); Men hvordan finder man ud af hvor mange parameter og hvilke man kan bruge til en funktion? Det gør man ved at kigge i dokumentationen til processing. Du finder alle funktioner i processings reference guide: processing.org/reference.



Figur 1: Højhat

Du kan bruge disse otte instruktioner for at kunne lave tegningen.

- size();
- line();

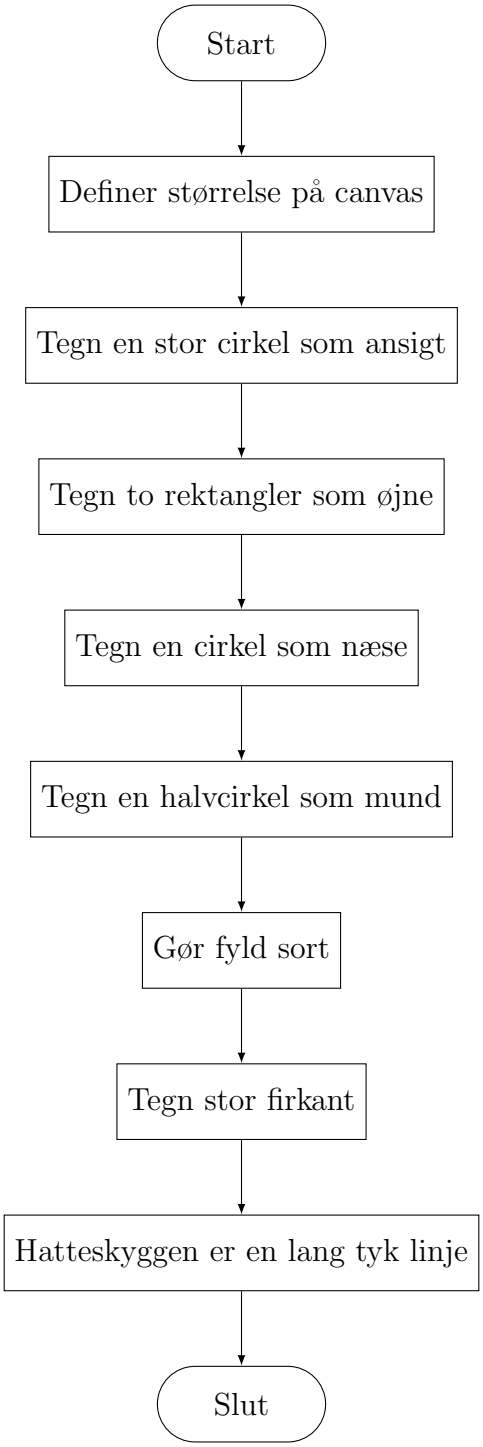
- strokeWeight();
- rect();
- square();
- circle();
- arc();
- fill();

Brug Processings dokumentation: processing.org/reference, for at finde ud af hvilke parameter de forskellige funktioner skal have.

- canvas (vindue som processing åbner) kan have størrelsen $400, 600$
- strokeWeight() er tykkelsen på stregen.
- fill() udfylder figuren med en RGB-farve.

Husk at koordinaterne $0, 0$ er øverste venstrehjørne (normalt vil det være nederste venstre) og er efter princippet: "hen ad vejen, ned til stegen". X,Y.

Brug rutediagrammet i figur 2 for at skrive koden:



Figur 2: Rutediagram

Hvad sker der, hvis du bytter om på rækkefølgen? Altså hvis du starter med øjne,næse og mund og så tegner ansigtet.

1.2 Opgave

Du har nu en fornemmelse af hvad sekventiel programmering er. Men en af styrkerne i Java er, at det understøtter produceral programmering. Hver procedure programmeres sekventielt. I Proccesing skal der altid være to procedurer. Setup og draw. Setup bruger vi til f.eks. at sætte størrelse på canvas eller andre initieringer. Draw er hoveddelen. Proceduren (vi kalder det også en funktion) looper 60 gange i sekundet. Vi kan selv definere alle de procedurer som vi har brug for.

Tilføj nu følgende linjer til dit program og flyt alt dit kode ned i proceduren/funktionen hoved.

```
void setup(){
  size(480, 120);
}

void draw(){
  head();
}

void head(){
  // Skriv din kode her. I mellem de to tuborg paranteser.
}
```

1.3 Opgave

Del dit program yderligere op. Således at én funktion tegner hatten. En anden tegner øjne, en tegner munden og den sidste tegner ansigtet. Kald funktionerne for: void hat(){} ,void eyes(){} ,void mouth(){}, void face(){}, husk at fjerne funktionen head(), når du er færdig.

1.4 Opgave

Gør nu dit canvas så stort at der er plads til to hoveder ved siden af hinanden. Tilføj to parametere til dine funktioner, en float x og en float y koordinat, udfra hvilke du kan tegne alle dine elementer af ansigtet. Ret dine linjer til så de tager udgangspunk i dine x og y koordinater. Er du rigtig god, kaler du dine funktioner med de samme værdier!

1.5 Opgave

Tegn figuren færdig med krop, arme og ben i hver deres funktion.

2 Primitiv Animation

Det vigtige i denne opgave er, at du skal lære at bruge dokumentationen til processing/Java. Herudover skal du bruge rutediagram til at strukturere din kode. Tænk på animation som stop motion teknik.

Du skal bruge nogle nye instruktioner

- frameRate()
- frameCount()
- noLoop()
- rotate()
- translate()
- popMatrix()
- pullMatrix()

Brug Processings dokumentation: processing.org/reference, for at finde ud af, hvad de forskellige funktioner gør og hvilke parameter de forskellige funktioner skal have.

2.1 Opgave

Lav et program, hvor et hjul, med minimum tre eger, ruller over skærmen fra venstre mod højre. Start med at lave et rute diagram.

2.2 Opgave

Udvid programmet så når hjulet forsvinder i højre side, dukker det op på den anden side igen.

2.3 Opgave

Lav programmet om så når hjulet rammer væggen, at det stopper og ruller tilbage hvor det kom fra.

2.4 Opgave

Lav et program som lave en primitiv animation af en mand som kan gå. Start med at lave et rute diagram.

Du kan finde inspiration i mine to eksempler på Github.

3 Løkker og Funktioner

Læs om while-løkker i kapitel 3.3 og om for-løkker i kapitel 3.4. Læs om funktioner i kapitel 3.5

Du kender nu til følgende datatyper og instruktioner:

Datatyper:

- float: kommatal.
- double: mere præcis end float.
- char: en enkelt karakter. Char er en unsigned byte og kan derfor ikke indeholde minustal.
- int: heltal.
- long: 2 gange så stor som en integer.
- boolean: sand eller falsk.

Instruktioner:

- size(); // sætter størrelsen for canvas
- line(); // tegner en linje
- stroke(); // farven på en streg
- strokeWeight(); // tykkelse af streg
- rect(); // tegner en rektangel
- circle(); // Tegner en cirkel
- arc(); // Tegner en bue
- fill(); // udfylder en figur med farve. [1]

[1] Find rgb farver her: www.rapidtables.com/web/color/RGB_Color.html

Vi skal også arbejde med funktioner. Du kender dem fra matematikken $f(x)$, hvor en funktion beregner/returnerer en værdi. Vi bruger funktioner når vi skal udføre den samme sekvens flere gange, med forskellige variabler. Funktioner i Java, består af 4 dele.

Navn: Funktioner skal navngives med et ikke reserveret ord (ord som er brugt i forvejen), men hvor navnet er sigende for funktionens funktion. I mit eksempel er navenet "minFunktion". Vi bruger navnet når vi skal bruge funktionen.

Returdatatype: Funktioner skal deklareres efter returdatatype. Funktionen kunne returnere en int, float, string eller char, men altid kun én ting. Hvis funktionen ikke returnerer noget, skal den defineres som void.

Parameter: Funktionen kan modtage en lang række forskellige parameter. En funktions paramenter skal angives i parantesen efternavenet samtidig med at vi deklarerer datatypen. Man kan deklarere mange parametere til sin funktion. Parameter er kun gyldige lokalt. Det vil sige at du kan ikke bruge en variabel du har deklareret i andre funktioner uden at skulle deklarere dem igen.

Kode: En funktions kode skal skrives imellem de to tuborgparanteser .

3.1 Opgave

Skriv et program hvor dit canvas er 800x800. Opret en funktion cirkel, som tegner en cirkel på dit canvas. Cirklen skal være rød. Ryd op efter dig. Navngiv din funktion så den fortæller hvad funktionen gør. Opret en funktion som tegner en firkant. Firkanten skal være blå. Navngiv din funktion så den fortæller hvad funktionen gør.0

3.2 Opgave

Tilpas din funktion så den tegner 8 cirkler. Brug et for loop og brug dit index i til at gange din x koordinat i cirklen, således at din cirkel ikke bliver tegnet det samme sted.

```
for (del 1; del 2; del 3) {
// Den sekvens af kode som løkken skal udføre
}
```

Et for-loop består af tre dele. Del 1 er en deklarering og initiering af vores index. Vi kalder index for i. Har vi brug for flere indlejret løkker følger vi

alfabetet og kalder den næste j så k etc. Del 2 er den betingelse som skal være opfyldt for at løkken bliver udført. Betingelsen knytter sig typisk til vores index i. For eksempel: i¡=10. Del tre beskriver den handling som skal ske med i efter hver iteration. Vi kan tælle i op med 1; i++. Eller trække en fra i–.

```
for (int i = 0; i < 10; i++){
  System.out.println(i);
}
```

Figur 3: For-løkke

3.3 Opgave

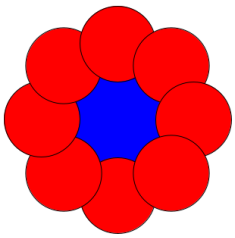
Tilpas din funktion firkant så den tegner et antal firkanter vertikalt. Hvor mange firkanter kan du få plads til?

3.4 Opgave

Omskriv din for-løkke til en while-løkke.

3.5 Opgave

Lave en blomst af cirkler. Placer en cirkel i midten og lav kronbladene af 8 cirkler. Du kan måske bruge funktionerne pushMatrix(), popMatrix, translate() og rotate()?



Figur 4: Blomst

4 Betingelser/Forgreninger

Læs om Betingelser/forgreninger i kapitel 3.2 i Systime bogen.

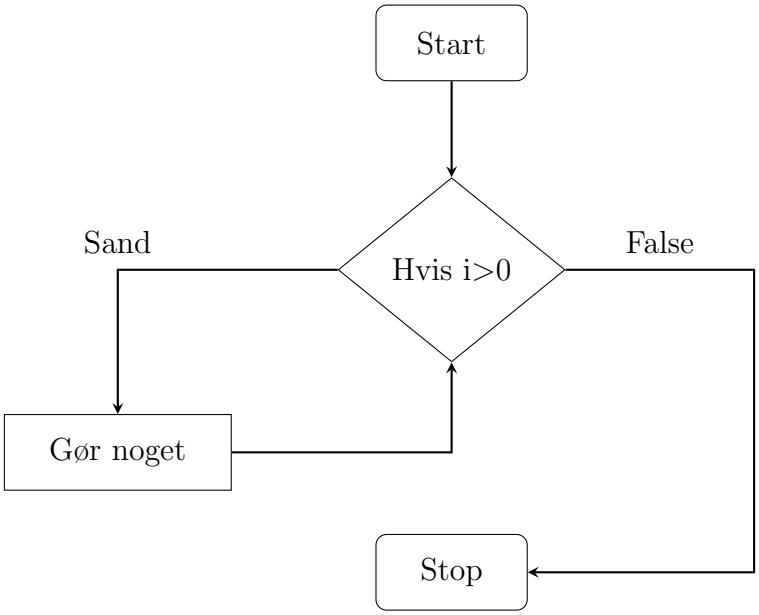
4.1 Operatorer til sammenligning

- Mindre end: $a < b$
- Mindre end eller lig med: $a <= b$
- Større end: $a > b$
- Større end eller lig med: $a >= b$
- Er lig med: $a == b$
- Forskellig fra: $a! = b$

4.2 Aritmetiske operatorer

- + Addition Lægger to værdier sammen. X+Y
- - Subtraktion Trækker to værdier fra hianden x - y
- * Multiplikation Ganger to værdier x * y
- / Division Dividerer en værdi med den anden x / y
- % Modulus Returnere resten ved division x
- ++ Inkrement Øger værdien af en variabel med 1 ++x
- – Dekrement Mindsker værdien af en variabel med 1 –x

```
if (betingelse) {
// Den sekvens af kode som løkken skal udføre hvis betingelsen er sand.
}
```



Figur 5: Rutediagram løkke

5 Ikke primitive datatyper

Vi skal i dette kapitel arbjede med ikke primitive datatyper. Det gælder for de primitive datatyper at de kan repræsenteres i en byte. Ikke primitive datatyper kræver meget mere plads. Primitive datatyper er foruddefineret i java, det er ikke primitive datatyper ikke, på nær String. Ikke primitive datatyper kan bruges til at kalde funktioner og bestemte operationer.

Den ikke primitive datatype String, er en række af karrakterer (Chars). Der er til datatypen knyttet en række metoder:

- toUpperCase() Gør alle karakterer til store bogstaver
- toLowerCase() Gør alle karakterer til små bogstaver
- substring() Returnerer en ny streng som er en del at den orginale streng.
- length() Returnerer en heltalsværdi som repræsenterer antallet af karakterer i strengen.
- indexOf() Returnerer indeks af den første forekomst af substreng i strengen.
- equals() Sammenligner to strenge
- charAt() Returnerer karakteren på den af index angivne plads.

Eksempler på andre ikke primitive datatyper: Strings, Arrays, Classes, Interface, etc.

5.1 Opgave

Lav opgaver på codingbat.com: String 1

6 Lektion 4 uge 43

Vi trækker håndbremsen, stopper op, og reflekterer over hvad vi har lært 'so far'.

Computere arbejder med 0,1 og alt basserer sig på det binæretalsystem. Det betyder at alt funktionalitet baserer sig på booskalgebra, AND, OR og NOT. De nøgleord bruger vi også når vi programmerer.

I har lavet en liste med ord og udtryk:

1. Instruktion

2. Sekvens
3. Funktion
4. Kontrolstruktur
5. Betingelser
6. Forgrening
7. Løkke
8. Funktion
9. Initiering
10. Deklaration
11. Parameter
12. Cammelback notation
13. Variabel (https://data-flair.training/blogs/java-data-types/)
    (a) Ikke primitive datatyper
        i. String
        ii. Array
        iii. klasser
        iv. Interfaces
    (b) Primitive datatyper
        i. Int
        ii. Float
        iii. Char
        iv. Boolean
        v. Byte
        vi. Short
        vii. long
        viii. Double.

6.1 Opgave 1

Denne opgave handler de forskellige datatyper. Til dette skal du opstille en tese (et vildt, men kompetent gæt) for min og max værdi af hver primitiv data type. Skriv et program, som kan beregne den maksimale værdi for en datatype. Vi kalder dette den induktive metode (specialtilfælde), fordi vi leder efter en special værdi (sort svane). Find evt. inspiration i programmet testDatatyper, som du finder på github. Noter alle dine resultater. Brug nu den deduktive metode (logiske), og beregn den maksimale værdi for hver primitiv datatype ud fra hvor meget plads der allokeres i compueterens hukommelse til datatypen. F.eks allokeres der (sjovt nok) en byte til datatypen byte. Du kan her finde svaret https://data-flair.training/blogs/java-data-types/ Noter alle dine resultater og slut af med at sammenholde din tese med resultatet af din induktive og deduktive metode og hvad der står i artiklen: https://data-flair.training/blogs/java-data-types/ Ekstra opgave: De to datatyper float og double er ikke lige nøjagtige. Det kan de se hved følgende opgave: Hvad giver kvadratroden af 2 gange med kvadratroden af 2? Lav et først et program med sqrt() som returnerer en float og herefter med Math.squrt() som returnerer en double. Forklar forskellen på de to funktioner og redegør for resultatet af de to instruktioner.

7 ROBO-Code

Formålet med dette forløb er at give den studerende en fornemmelse af hvad OOP er. Vi nedarver funktioner og bruger klassens metoder og attributter. Det er en indledning til næste forløb som er OOP.

De studerende skal programmere en robot til at skyde andre robotter på en lukket bane og undgå selv at blive ramt. Der kan vælges mellem følgende strategier:

- Offensiv strategi - en agressiv robot som er opsøgende og konfronterende.
- Deffensiv strategi - en passiv robot som er forsvarende og undvigende.
- En kombination af overstående basseret på hvor mange robotter som er tilbage i spillet.

7.1 En robots egenskaber

Opret en ny junior robot. Du skal nu overveje 5 ting.

- Hvordan skal vi initiere robotten? Skal den køre et sted hen?
- Hvordan skal din robot bevæge sig på banen?
- Hvad skal der ske når din robot opdager (scanner) en fjende?
- Hvad skal der ske når din robot bliver ramt?
- Hvad skal der ske hvis din robot rammer væggen?

Et eksempel: Så længe at din robot ikke har opdaget en fjende, er blevet ramt eller er kørt ind i en væg, hvordan skal den så bevæge sig på banen? Jeg har lavet et eksempel hvor robotten kører 100 frem drejer kanonen 360 grader, kører 100 tilbage og drejer 360 grader.

```
public void run() {
  // Initialization of the robot

  // Some color codes: blue, yellow, black,
  // white, red, pink, brown, grey, orange...
  // Sets these colors (robot parts):
  // body, gun, radar, bullet, scan_arc
  setColors(orange, blue, white, yellow, black);

  // Robot main loop
  while(true) {
    // Replace the next 4 lines
    // with any behavior you would like
    ahead(100);
    turnGunRight(360);
    back(100);
    turnGunRight(360);
  }
}
```

7.2 Opgave

Min robot kører frem og så drejer. Men der findes en kommando som kører frem og drejer i samme bevægelse. Både frem og tilbage. Brug dokumentationen, `https://robocode.sourceforge.io/docs/robocode/robocode/JuniorRobot.html`, til at finde ud af hvilke parameter du skal tilføje til funktionen turnAheadLeft() eller turnBackRight().

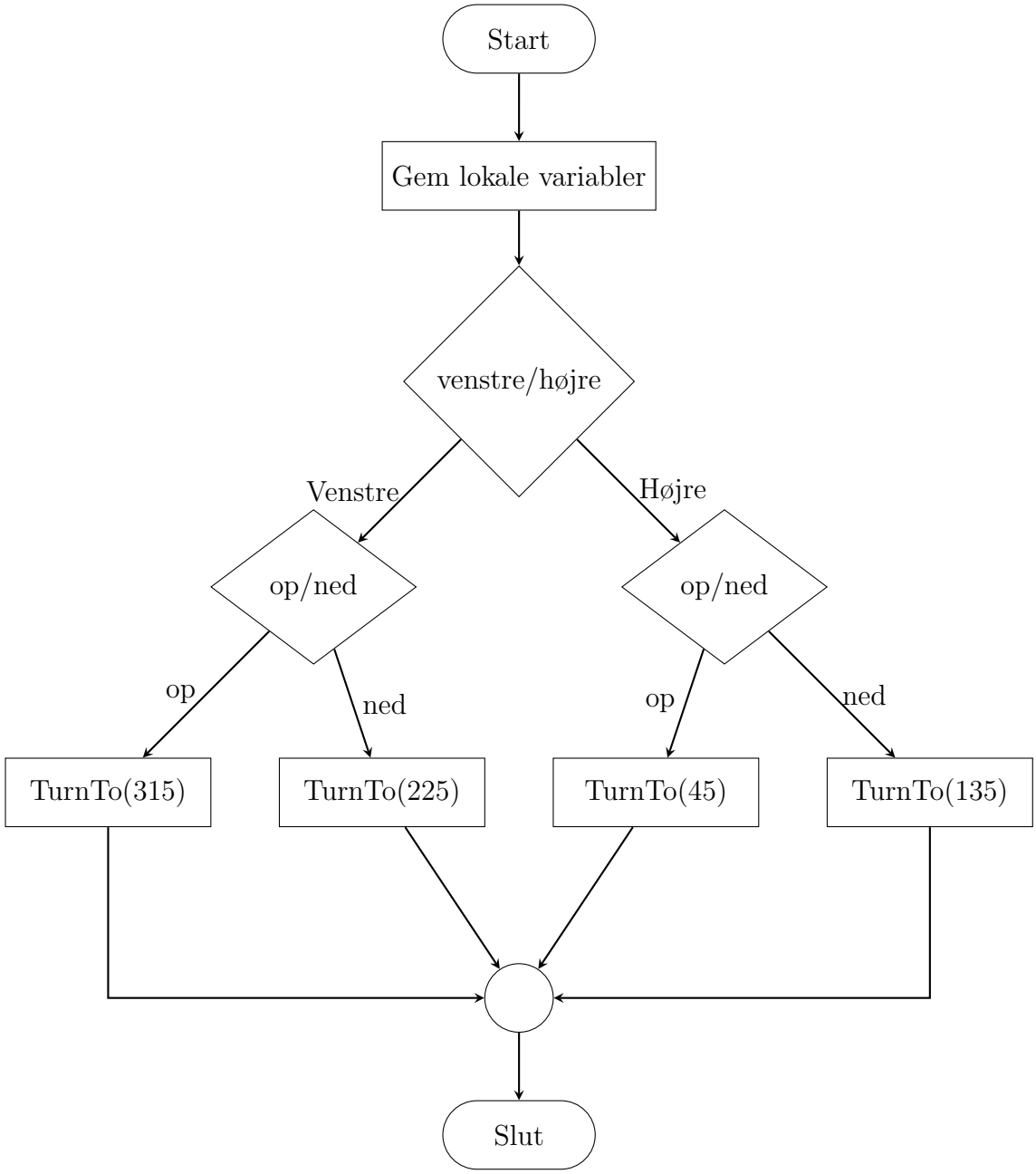
Når spillet starter kunne en stategi være at søge i mod den nærmeste væg. Men vi ved ikke i hvilken retning robotten peger eller hvilken væg som er nærmest. Men udfra vores robots x værdi kan vi vide om vi skal mod højre eller venstre. Er x mindre end fieldWidth/2 er der kortere vej mod venstre end der er til højre. Vi kan bruge den samme metode til y.

> Koordinatetsystemet i robocode har 0,0 i nederste venstre hjørne.

- public int robotX - er robottens x-position
- public int robotY - er robottens y-position

- public int fieldWidth - er banens brede på x-axelen
- public int fieldHeight - er banens brede på y-axelen
- public int heading - retning i grader som robotten peget i

Skriv koden, som får din robot til at søge i mod den nærmeste væg. Brug evt. rutediagrammet herunder. Placer din kode i under linjen: // Initialization of the robot should be put here



Figur 6: robot init

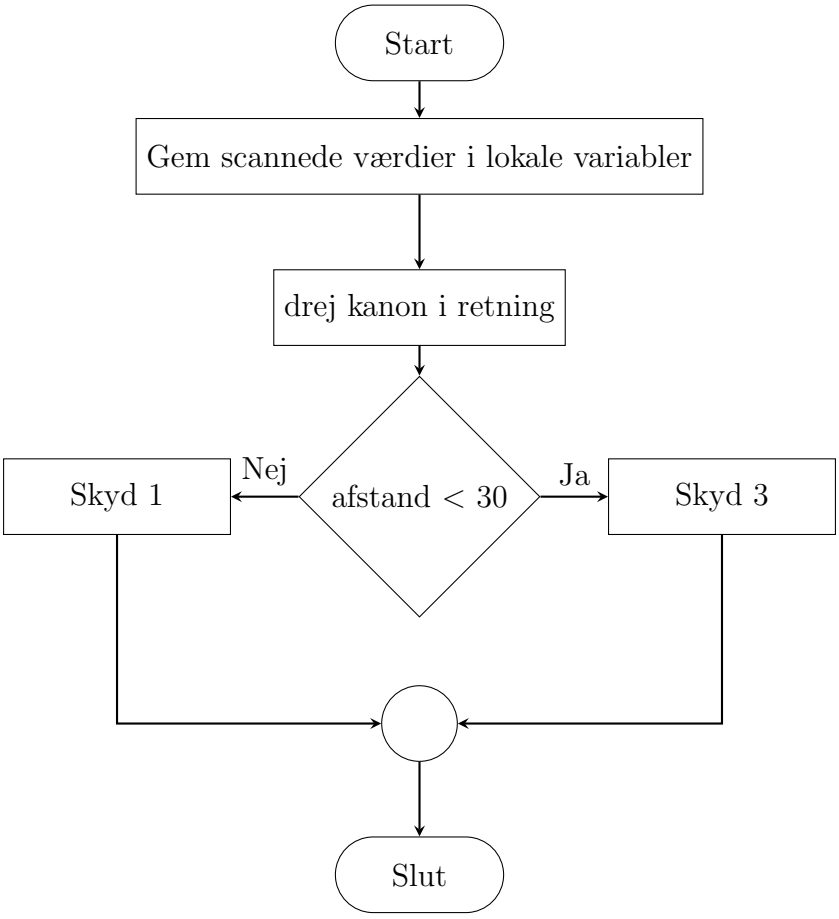
7.3 Opgave

Når vi drejer kanonen modtager vi en række data fra vores scanner.

- public int scannedDistance - afstand til den nærmeste fjende
- public int scannedAngle - som et udtryk for grader i forhold til verdenshjørner
- public int scannedBearing - som et udtryk for grader i forhold til din egen robot
- public int scannedEnergy - den mængde energi fjenden har tilbage
- public int scannedVelocity - hastigheden
- public int scannedHeading - retning som robotten bevæger sig i

Det er information som vi kan bruge til at beslutte om vi vil forsøge at nedkæmpe målet. Robotterne bevæger sig, og hvis vi skyder efter en robot som er for langt væk risikerer vi at robotten er kørt når vores projektil når frem.
Når spillet starter har vi 100 energi. Vi kan affyre vores kanon med 0.1 til 3 i energi. Vi modtager kun energi når vi rammer fjenden. Derfor kunne en strategi være at skyde en lille mængde energi af, når modstanderen er lagt væk og mere energi når modstanderen er tættere på og chancen for at ramme er højere.

Skriv koden til onScannedRobot() hvor du tager højde for hvor langt modstanderen er væk. Brug evt rutediagrammet her under. Udvid eventuelt koden så du varierer skudets energi endnu mere i forhold til afstanden. Lav et rutediagram som beskriver din kode.



Figur 7: onScannedRobot()

7.4 Opgave

Lav en strategi for hvad der skal ske hvis du kører ind i væggen. En strategi kunne være at bakke tilbage dreje og så fortsætte. Men problemet er, hvis vi rammer vægen med robottens højreside og drejer til højre skal vi dreje mere for at komme fri af vægen end hvis vi drejer til venstre. Det tager længere tid og gør os sårbare overfor at blive ramt.

- hitWallAngle - angiver de 4 verdenshjørner. 0=nord, 90=øst 180 = syd og 270 er vest
- hitWallBearing - angiver antal grader i forhold til dig køretøj

Rammer du en væg og hitWallBearing værdien er -5, betyder det at din robots venstre side er kørt i muren i en vinkel af 5 grader og visa versa.

Start med at lave et rutediagram.

7.5 Opgave

Lav en strategi for hvad der skal ske hvis du bliver ramt af et projektil.

- public int hitByBulletAngle, giver dig retningen hvorfra der er skudt
- Public int hitByBulletBearing giver dig retningen set i forhold til din robots retning.

En strategi kunne være:

- Skyde tilbage i samme retning som skudet kom fra
- At scanne i retningen af hvor skuddet kom fra, for at få en mere nøjagtig position
- At flytte sig væk fra retningen hvor skudet kom fra

7.6 Opgave

Tilføj en action event: public void onHitRobot() og lav en strategi for hvad du vil gøre hvis du rammer en modstander.

En strategi kunne være:

- Skyde igen, men denne gang med mere energi
- Køre tættere på hvis modstanderens energi er lav. brug public int scannedEnergy, for at kende modstanders energi niveau
- Flygt, fjerne dig fra angriberen
- Forfølg modstanderen hvis denne flygter fra dig
- En kombination af overstående

Der er mange muligheder, så du kan prøve at eksperimentere med forskellige taktikker for at se, hvad der virker bedst for dig.

Brug evt. et rutediagram.

7.7 Den svære opgave

Du skal lave to robotter.
Robot nummer 1, skal placere sig i fieldHeight/2 og køre frem og tilbage over skærmen. Denne robot er dit mål.
Lav en robot nummer 2, som skal placere sig i midten af bunden af banen og skyde efter robot 1.
Hvor mange skud skal du bruge for at nedkæmpe målet ?