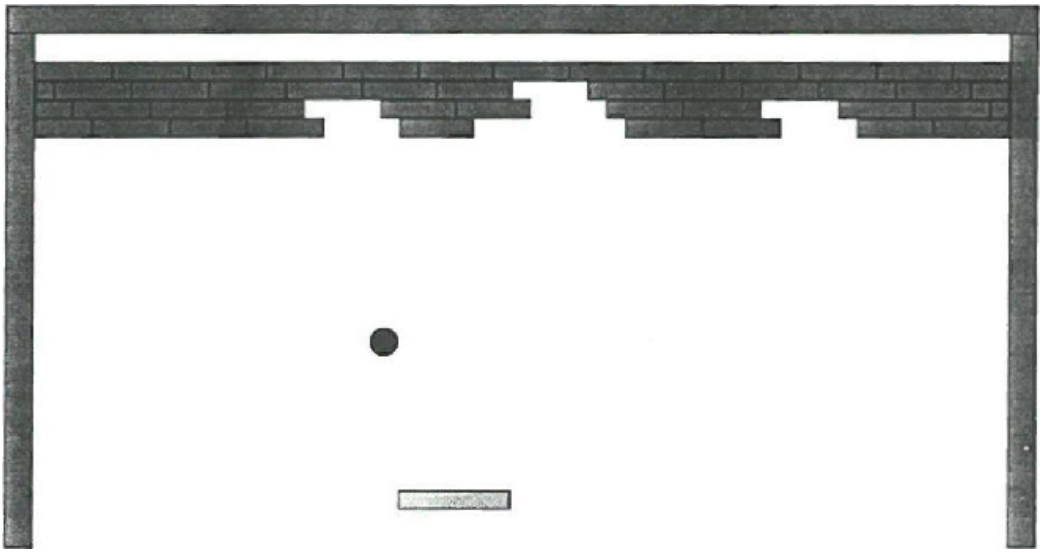
Strukturel ProgramUdvikling



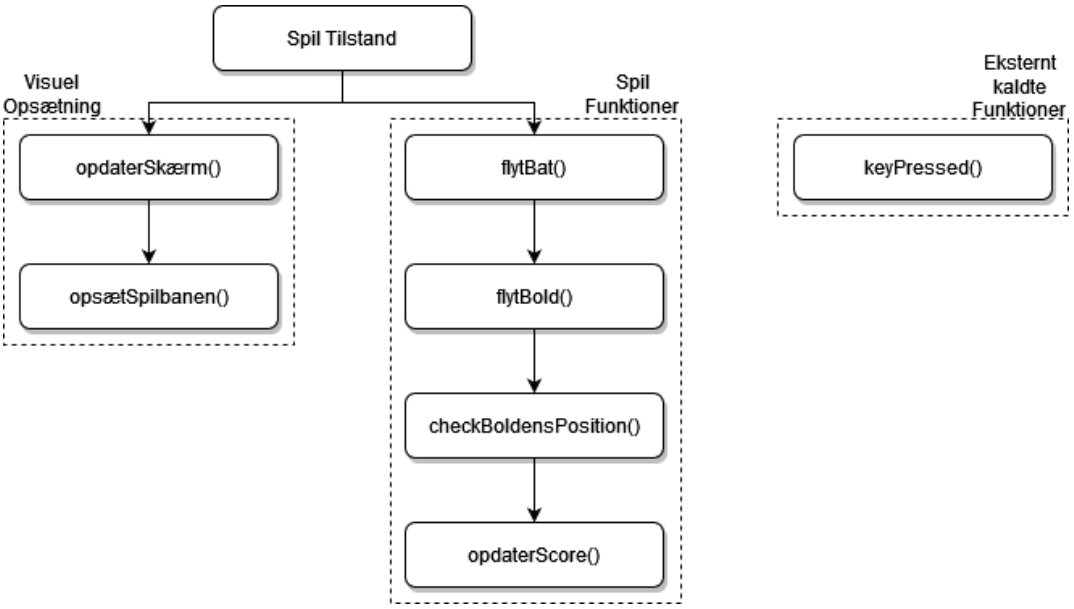
For at forklare strukturel programmering aka. SPU (Engelsk: procedural programming) vil vi kigge på designet af et simpelt video spil - Breakout (Der er 1 spiller version af spillet pong).

For dem der ikke kender dette spil, så består det af en bane som illustreret ovenfor, hvor der er en bold som man bruger til at slå hul i en mur (de fire rækker øverst på banen). Bolden slår en sten ud af muren hver gang den rammer en sten, og skifter derefter retning efter princippet udfaldsvinkel = indfaldsvinkel (Hvilket kendes fra fysik). Når bolden rammer en af de tre sider (højre, venstre, og øverst) reflekteres bolden også. Nederst er et bat som man skal fange bolden med for dermed at reflektere den op mod muren igen. Battet kan man styre.
Allerede her er udviklingen i gang, men der tales her om en før programmering udvikling. Og her laves ofte i denne rækkefølge:

1. Use cases (Eksempel i bilag).
2. Kravspecifikation (Eksempel i bilag).
3. Tilstandsdiagrammer (Eksempel i bilag). Husk kan også bruges til dokumentation.
4. Flow diagram (Overordnet) eller mere detaljeret pseudokode, der viser funktionaliteten fra brugerens synspunkt (Eksempel af spil tilstanden i bilag). Husk flowdiagram kan senere udvides og bruges som dokumentation. Det kan pseudokode *IKKE*!

Strukturel programmering

Lad os starte med at pointere at dette er det 5 punkt i metodes.
Ved strukturel programmering vil vi starte med at lave et videospil ved at fokusere på hvad der skal udføres. Ovenstående viser en typisk måde at foretage en strukturel nedbrydning af spillet på. Man starter med en meget generel procedure (hele programmet er her set som en procedure). Derefter kigger man på hvilke enkelt operationer denne består af. I eksemplet ovenfor består den af en procedure der tegner banen, og dernæst kaldes en procedure som spiller et enkelt spil.

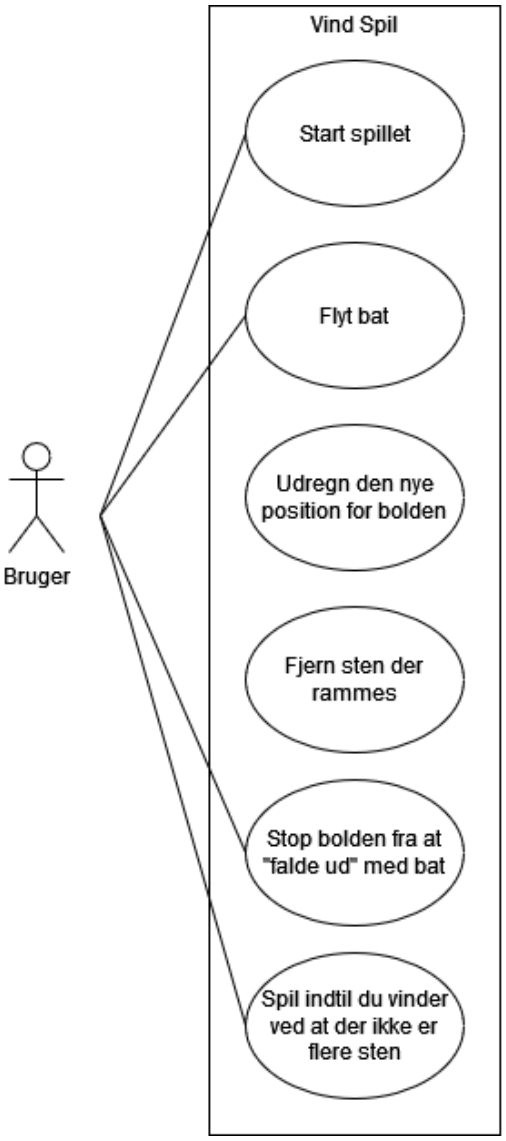


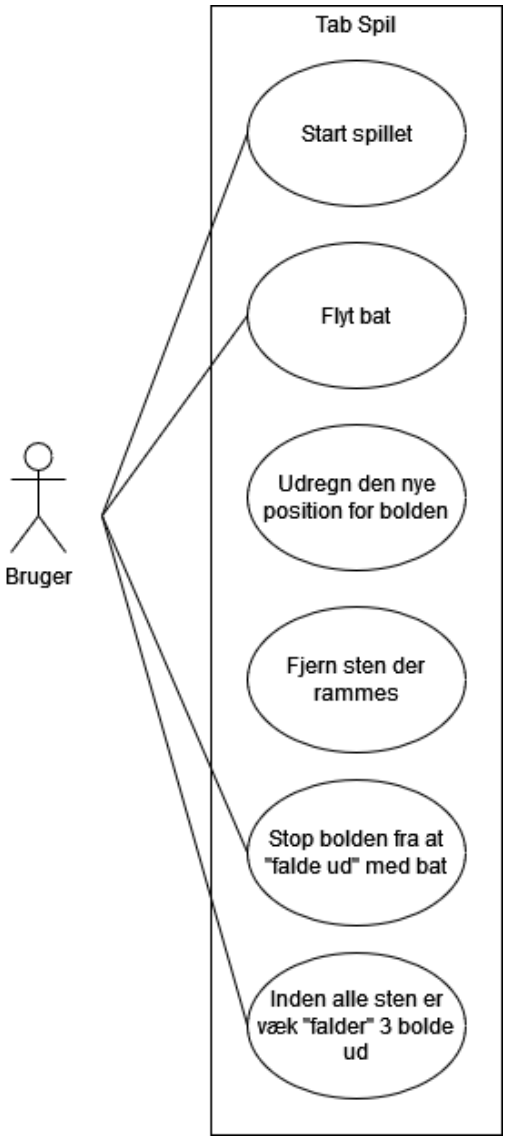
Disse to procedurer kan så igen brydes ned i under procedurer. Program udviklingen i strukturel programmering består altså af følgende trin:

1. Given en procedure A.
2. Hvis proceduren A har en direkte implementation er vi færdige.
3. Ellers nedbrydes A i en række under opgaver, som hver gøres til procedurer.
4. For hver under procedure gentages punkt 1-4.

Bilag - Use cases

Skriv use cases til de mest brugte situationer du vil bruge programmet til. Use cases kan enten være en fortælling eller et diagram. Jeg bruger i dette eksempel diagrammer til at vise hvordan man “vinder” og “taber” spillet.

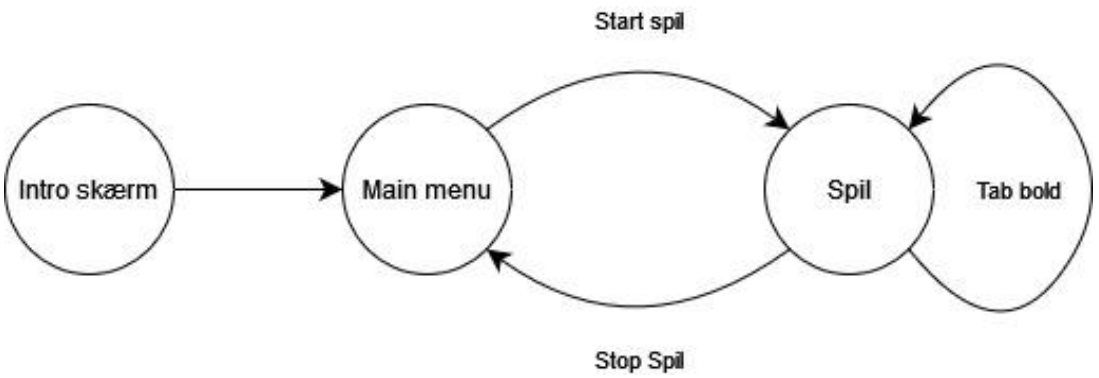




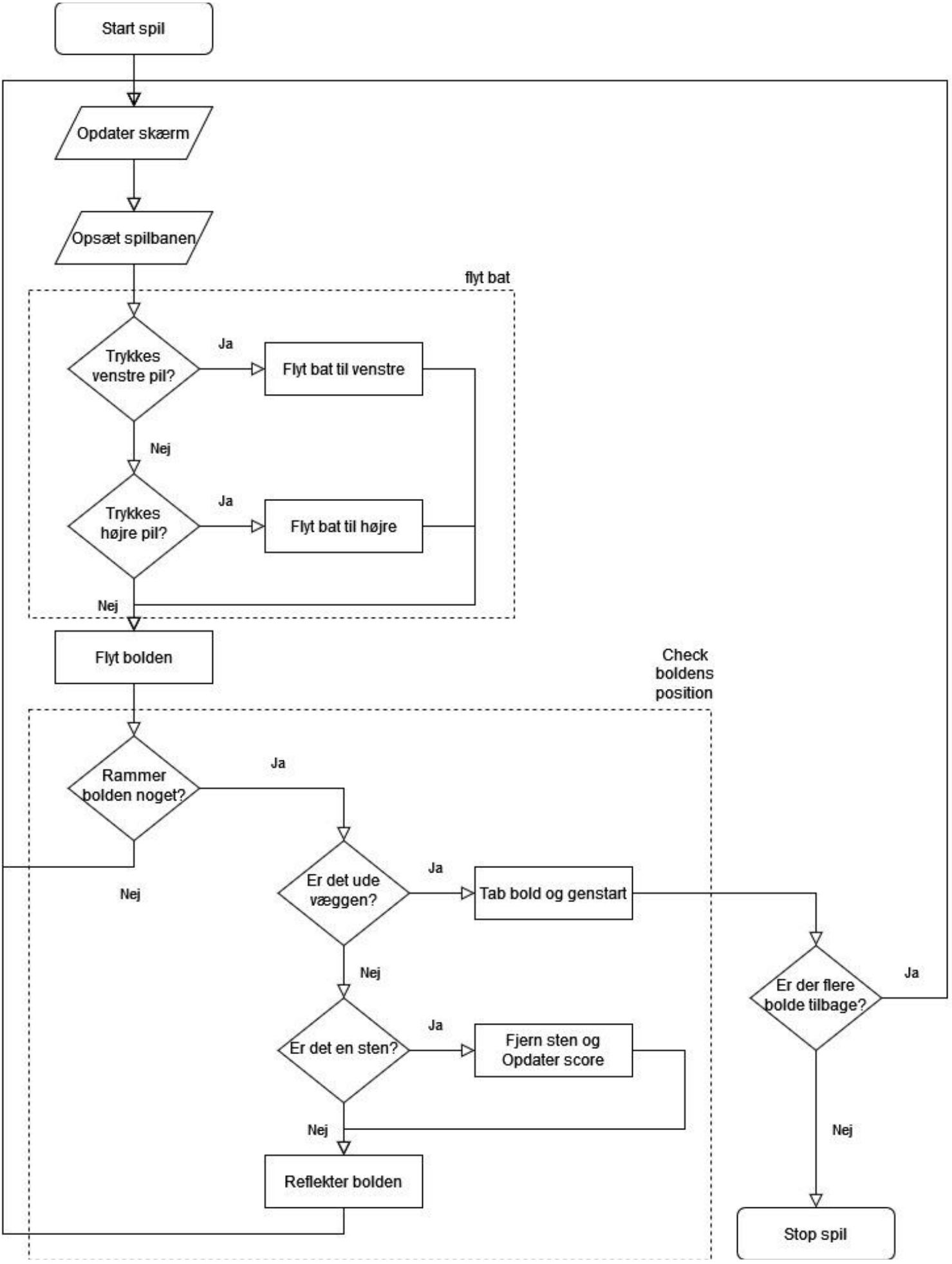
Bilag - Kravspecifikation

Krav	Test
Det skal være muligt at vinde ved at fjerne alle sten	#1 Spil spillet indtil alle sten på nær en i første række er fjernet og fjern den herefter. #2 Spil spillet indtil alle sten på nær en i anden række er fjernet og fjern den herefter. osv.
Spillet fortsætter indtil alle bolde er tabt	#1 Tab alle bolde uden at ramme dem. #2 Tab de to første bolde i starten uden at ramme dem, men tab den sidste ved sidste sten. #3 Tab den første bold i starten uden at ramme dem, men tab de sidste ved sidste sten.
osv.	osv.

Bilag - Breakout tilstandsdiagram



Bilag - Breakout flowdiagram for Spil tilstanden



Bilag - Pseudokode for spil tilstanden

Overordnet	Mere detaljeret

Opsæt skærm
loop så længe liv >0 {
Flyt bold
Flyt battet
Opdater skærm
Check boldens position
Opdater score
}
	Opsæt skærm
	loop så længe liv >0 {
	Flyt bold
	Er der trykket på en kontrolknap?
	Flyt battet tilsvarende
	Opdater skærm
	Rammer bolden et element?
	Hvis sten fjern sten & Opdater score
	Hvis sten rammer ude væggen, så genstart & liv--
	Udregn boldens refleksion
	}

Bilag - Dokumentation af kode (Kommentarer i koden)

Ud over de 2 nævnte metoder til dokumentation, så er der også kommentarer i koden. Der er 2 typer af kommentarer, hvor den første er Program & funktion (Metode) kommentarer og den anden er kommandolinje kommentarer.

Program & funktion kommentarer

Overfladiske kommentarer	Detaljerede kommentarer

/*
Breakout Eksempel Version 0.01 :)
*/
 	
//Lav globale variabler
Boolean TilstandIntroTilstand=true,
TilstandMainMenuTilstand=false,
TilstandSpil=false;
...
/*
Sæt op Program
*/
void **setup**() {
	/*
	Breakout Eksempel Version 0.01 :)
	
	Input Keyboard:
	Venstre pil flytter bat til venstre
	Højre pil flytter bat til højre
	Op pil lægger 1 til hastigheden
	Ned pil trækker 1 til hastigheden (Går den under 0, så skifter den retning)
	*/
	
	//Lav globale variabler
	Boolean TilstandIntroTilstand=true, TilstandMainMenuTilstand=false, TilstandSpil=false; //De tre tilstande fra tilstandsdiagrammet.
	...
	/*
	1) Sæt op Program
	2) Vis Intro skærm
	3) Sæt op globale variabler til at passe til skærmopløsning
	4) Start spillet
	*/
	void **setup**() {

Kommandolinje kommentarer

Overfladiske kommandolinje kommentarer	Detaljerede kommandolinje kommentarer
void **keyPressed**() {	void **keyPressed**() {

```
  if (key == CODED) {
    if (keyCode == RIGHT) {       //Flyt bat
      FlytBat(true);
    } else if (keyCode == LEFT) {  //Flyt bat
      FlytBat(false);
    }
    if (keyCode == DOWN) {    //Sænk bold
hastighed
      float temp = BoldensHastighed.mag();
      temp--;
      BoldensHastighed.normalize().mult(temp);
    } else if (keyCode == UP) { //Øg bold hastighed
      float temp = BoldensHastighed.mag();
      temp++;
      BoldensHastighed.normalize().mult(temp);
    }
  }
}
```

```
  if (key == CODED) {
    if (keyCode == RIGHT) {       //Flyt bat
      FlytBat(true);          //Kald FlytBat() med true = højre
    } else if (keyCode == LEFT) { //Flyt bat
      FlytBat(false);         //Kald FlytBat() med falsk =
venstre
    }
    if (keyCode == DOWN) {            //Sænk bold hastighed
      float temp = BoldensHastighed.mag(); //Gem
hastigheden i en midlertidlig lokal variabel
      temp--;                         //Minimer hastigheden med
1
      BoldensHastighed.normalize().mult(temp);
//Opdater hastigheden
    } else if (keyCode == UP) {               //Øg bold
hastighed
      float temp = BoldensHastighed.mag();
//Gem hastigheden i en midlertidlig lokal variabel
      temp++;                               //Øg hastigheden med
1
      BoldensHastighed.normalize().mult(temp);
//Opdater hastigheden
    }
  }
}
```

Råd til kommentarer:

- De skal hjælpe med at forstå opbygningen af koden
- Til funktioner er formål, input, return og effekt (Ændres globale variabler?) vigtige kommentarer
- Vælg det niveau til kommentarer, der passer bedst til dig, og samtidigt opfylder de andre råd
- Kommentarer skal kunne forstås af både dig og andre som der ser eller arbejder med dig
- Lav kommentarerne mens du skriver koden (Især Program & funktion kommentarer, der hjælper med struktur. Hvis du har svært ved at lave dem, så bør du overveje om du er parat til at skrive kode)
- Hovedreglen til dokumentation er at man altid har lavet for lidt

Struktur til programmet i praksis

```
/*
  Breakout Eksempel Version 0.01 :)

 Input Keyboard:
 Venstre pil flytter bat til venstre
 Højre pil flytter bat til højre
 Op pil lægger 1 til hastigheden
 Ned pil trækker 1 til hastigheden (Går den under 0, så skifter den retning)
 */

//Lav globale variabler
Boolean TilstandIntroTilstand=true, TilstandMainMenuTilstand=false, TilstandSpil=false; //De tre tilstande fra tilstandsdiagrammet.
Byte SideTykkelse = 10, BatWidth = 77, BoldSize = 8, LivTilbage = 3, BatHastighed = 15;
int BatPositionX = 3*BatWidth;
PVector BoldensHastighed = new PVector(random(-1, 1), random(-1, 0)), BoldensPosition = new PVector(3*SideTykkelse, 3*SideTykkelse);
```

```
color BackGroundColour = color(255, 204, 0), BatColour = color(0, 0, 0), BoldColour = color(255, 255, 255), VaegColour = color(255, 255, 255);

/*
 Sæt op Program
 Vis Intro skærm
 Sæt op globale variabler til at passe til skærmopløsning
 Start spillet
*/
void setup() {
  fullScreen(2);  //Programmet er udviklet på et 2 skærm setup, og her vælges skærm 2
  noStroke();
  ellipseMode(CENTER);
  rectMode(CENTER);
  if (TilstandIntroTilstand) {
    //Tegn intro side og pause i 20 sekunder
    TilstandIntroTilstand=false;
  }
  BatPositionX = width/2;
  BoldensPosition = new PVector(width/2, height-3*SideTykkelse);
  opsaetSpilbanen();
  TilstandSpil=true;
}

void draw() {
  if (TilstandSpil) {
    flytBold();
    opdaterSkaerm();
    if (LivTilbage <= 0) {
      TilstandSpil=false;
    }
  }
}

void keyPressed() {
  if (key == CODED) {
    if (keyCode == RIGHT) {       //Flyt bat
      flytBat(true);              //Kald Funktionen FlytBat med falsk = højre
    } else if (keyCode == LEFT) {  //Flyt bat
      flytBat(false);             //Kald Funktionen FlytBat med falsk = venstre
    }
    if (keyCode == DOWN) {                //Sænk bold hastighed
      float temp = BoldensHastighed.mag();  //Gem hastigheden i en midlertidlig lokal variabel
      temp--;                             //Minimer hastigheden med 1
      BoldensHastighed.normalize().mult(temp);  //Opdater hastigheden
    } else if (keyCode == UP) {           //Øg bold hastighed
      float temp = BoldensHastighed.mag();  //Gem hastigheden i en midlertidlig lokal variabel
      temp++;                             //Øg hastigheden med 1
      BoldensHastighed.normalize().mult(temp);  //Opdater hastigheden
      println(BoldensHastighed);
    }
  }
}

/*
```

```
 Sæt op banen med vægge og baggrund
*/
void opsaetSpilbanen() {
  background(BackGroundColour);
  fill(VaegColour);
  rect(width/2, SideTykkelse/2, width, SideTykkelse);
  rect(SideTykkelse/2, height/2, SideTykkelse, height);
  rect(width-SideTykkelse/2, height/2, SideTykkelse, height);
}

/*
 Undersøg om boldensposition har en effekt
*/
void checkBoldensPosition() {
  if (BoldensPosition.x>width-SideTykkelse) {
    BoldensPosition.x=width-SideTykkelse;
    BoldensHastighed.x=-1*abs(BoldensHastighed.x);
  } else if (BoldensPosition.x<SideTykkelse) {
    BoldensPosition.x=SideTykkelse;
    BoldensHastighed.x=abs(BoldensHastighed.x);
  }
  if (BoldensPosition.y<SideTykkelse) {
    BoldensPosition.y=SideTykkelse;
    BoldensHastighed.y=abs(BoldensHastighed.y);
  } else if (BoldensPosition.y>height) {
    BoldensPosition = new PVector(width/2, height-3*SideTykkelse);
    BoldensHastighed = new PVector(SideTykkelse, -SideTykkelse);
    LivTilbage--;
  }
  if (BoldensPosition.y>=height-2*SideTykkelse) {
    if (BoldensPosition.x<BatPositionX+BatWidth/2 && BoldensPosition.x>BatPositionX-BatWidth/2 ) {
      BoldensHastighed.y=-1*abs(BoldensHastighed.y);
    }
  }
}

/*
 Hold styr på points
*/
void opdaterScore() {
}

/*
 Sæt op spillets skærm
*/
void opdaterSkaerm() {
  opsaetSpilbanen();
  fill(BoldColour);
  circle(BoldensPosition.x, BoldensPosition.y, BoldSize);
  fill(BatColour);
  rect(BatPositionX, height-1.5*SideTykkelse, BatWidth, SideTykkelse);
}

/*
 Udregn boldens nye position
```

```
*/
void flytBold() {
  BoldensPosition.add(BoldensHastighed);
  checkBoldensPosition();
}

/*
 Udregn battets nye position
*/
void flytBat(boolean retning) {
  if (retning) {
    BatPositionX+=BatHastighed;
  } else {
    BatPositionX-=BatHastighed;
  }
}
```