# 2048 i pygame

## Step 0: Lav et vindue

Inden vi rigtigt kan begynde, skal vi lige have det basale på plads. Før vi kan lave en masse sej grafik (eller firkanter i vores tilfælde) skal vi have et sted at tegne det på. Koden nedenunder er vores pygame ”opstarts”-fil, hvilken indeholder al koden nødvendig til at få et vindue op man så kan lukke igen.

import pygame

højde = 500

bredde = 500

skærm = pygame.display.set\_mode((bredde, højde))

def **tegn**():

skærm.fill((0,0,0))

*#TEGN ET ELLER ANDET*

pygame.display.flip()

kør = True

while kør:

tegn()

for event in pygame.event.get():

if event.type == pygame.QUIT:

kør = False

Tjek om der kommer et vindue med en sort baggrund op, der kan lukkes ved at trykke på krydset.

## Step 1: Hvad er 2048 og hvordan laver vi det?

Som vi også snakkede om sidste gang, er det første man skal når man vil lave et spil er at finde ud af hvad man egentlig gerne vil lave. Selvom vi ”bare” laver en kopi af noget allerede eksisterende, er dette trin stadig rigtig vigtigt. For hvad er 2048 egentlig? Sidst fik vi det snævret ind til: ”Bræt med talklodser, der kan bevæge sig”. Det er selvfølgelig lidt mere kompliceret end bare som så, men det giver os en ide om hvordan vi kan dele problemet op i mindre bidder, der er nemmere at gå til. Så lad os starte med den første del af sætningen ”Bræt med talklodser, …”. Hvordan kan vi lave sådan et i vores kode? Det snakkede vi allerede lidt om sidste gang, men nu tager vi den lige igen for prins Knud. Da vi har et bræt med fire rækker og fire kolonner, hvor der i hver celle er et tal skal vi have noget kode som har nogenlunde samme struktur, hvilket er lister. Ikke bare lister, men lister inden i lister! Så vores bræt kommer til at være fire lister inden i en større liste, hvor der i hver af de små lister kommer til at være de fire kolonner af tal. Når et felt er tomt er der bare et nul i. Tilføj nedenstående kode lige under, hvor vi laver skærmen i vores program.

bræt = [[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]]

Kan det køre så fortsæt ellers få noget hjælp.

## Step 2.1: Kan i se brættet?

I foregående trin lavede vi selve brættet i koden, men det kan man kun se hvis man rent faktisk kigger i koden. Lad os prøve at få tegnet brættet på skærmen, hvilket egentlig bare betyder lad os få tegnet alle klodserne på skærmen. For at gøre det laver vi en tegnBræt funktion, hvori vi tegner alle klodserne som firkanter. I stedet for at skrive funktionen til at tegne en firkant 4\*4 gange kan vi gøre det nemmere for os selv ved at få computeren til at gentage tegn firkantfunktionen. Men vi vil ikke bare have at den tegnes det samme sted hver gang så vi skal have noget, der holder styr på hvilken kolonne og række vi er i. Derfor bruger vi ikke bare en, men hele to for-løkker. Hvor den ene for-løkker tæller fra række 0 til 3 og den anden tæller fra kolonne 0 til 3. Med det kan vi lave noget magisk matematik for at finde de rigtige koordinater. For at finde de rigtige koordinater skal vi også vide, hvor brede vores klodser skal være, hvilket vi gemmer i to variabler. Nedenunder har vi vores tegnBræt funktion prøv og se om du ikke kan beregne x- og y-koordinatet når vi ved hvilken klods vi er på (hvilket vi ved ud fra variablerne rækkeI og klodsI).

klodsBredde = bredde // 4

klodsHøjde = højde//4

def **tegnBræt**():

for rækkeI in **range**(4):

for klodsI in **range**(4):

pygame.draw.rect(skærm, (255, 0, 0), (x, y, klodsBredde, klodsHøjde))

Når du har fundet ud af hvordan man beregner x- og y-koordinatet skal du bare kalde tegnBræt() funktionen i din tegn funktion imellem de skærm.fill og pygame.display.flip.

Hvad ser du og hvorfor ser det sådan ud?

## Step 2.2: Sidder de ikke lidt tæt?

Hvis I i forrige delstep endte med et tegn firkant funktion som så omtrent sådan her ud har I gjort det godt.

pygame.draw.rect(skærm, (255, 0, 0), (klodsI\*klodsBredde, rækkeI\*klodsHøjde, klodsBredde, klodsHøjde))

Men nu ligner det bare lidt at vi har ændret baggrundsfarven, hvilket jo ikke helt er det vi vil have, vi vil nemlig gerne have individuelle klodser. Måden vi kan få det på er ved at lægge lidt afstand imellem klodserne. Hvilket skal forstås ret bogstaveligt vi lægger bare lidt til positionen og trækker det dobbelte fra bredden og højden og så har vi fået noget plads imellem dem. Men vores tegn firkant funktionskald er ved at blive ret langt, så nu er det tid til at dele det op i flere funktioner! Så nu kan vi lave en funktion, der hedder tegnKlods som så tegner pågældende klods. Den skal vide hvilken klods der er tale om, altså rækkeI og klodsI. I tegnKlods funktionen kan vi nu også gøre tegnFirkant funktionen kortere ved at lave x- og y-koordinatet til variabler.

klodsBredde = bredde // 4

klodsHøjde = højde//4

def **tegnBræt**():

for rækkeI in **range**(4):

for klodsI in **range**(4):

tegnKlods(rækkeI, klodsI)

def **tegnKlods**(rækkeI, klodsI):

x = klodsI\*klodsBredde

y = rækkeI\*klodsHøjde

pygame.draw.rect(skærm, (255, 0, 0), (x, y, klodsBredde, klodsHøjde))

Tjek om det virker nu hvor vi har delt det op. Hvis ja så fortsæt, ellers spørg om hjælp (prøv din side-pirat).

Nu kan vi så tilføje afstanden. Vi kan ikke på forhånd sige præcis hvor meget afstand vi gerne vil have og det kan også ændre sig hen ad vejen, derfor skal vi bruge en variabel. Så vi laver en afstands variabel, hvilken vi så lægger til vores x- og y-koordinat og trækker fra (to gange) i højden og bredden. Nu er vores højde og bredde også pludseligt blevet ret komplekse så lad os også hive dem ud i hver deres variabel.

Afstand = 5

def **tegnKlods**(rækkeI, klodsI):

x = klodsBredde \* klodsI + afstand

y = klodsHøjde \* rækkeI + afstand

b = klodsBredde - 2\*afstand

h = klodsHøjde - 2\*afstand

pygame.draw.rect(skærm, (255, 0, 0), (x, y, b, h))

Kan man nu se at der er tale om forskellige klodser? Er det nok afstand, er det for meget, prøv at eksperimenterer.

## Step 2.3: Men hvilket tal er der tale om?

Nu hvor vi har fået lavet nogle individuelle klodser vil vi også gerne kunne se hvilket tal der er indeni dem. For at gøre det skal vi jo have gang i at tegne noget tekst (selvom det egentlig er tal). Derfor skal vi først hevet en font ind i pygame. For at gøre det skal vi skrive nedenstående lige neden under vores skærmvariabel.

pygame.font.init()

font = pygame.font.SysFont(None, 30)

Nu kan vi så lave vores tekstbillede i tegnKlods funktionen og vise det på skærmen. Teksten vi gerne vil vise er jo tallet, hvilket vi finder ved at slå op i brættet, hvor vi får et tal tilbage vi skal lave om til tekst, hvilket vi gør med str() funktionen. Så vores tegnKlods funktion ender med at se således ud:

def **tegnKlods**(rækkeI, klodsI):

x = klodsBredde \* klodsI + afstand

y = klodsHøjde \* rækkeI + afstand

b = klodsBredde - 2\*afstand

h = klodsHøjde - 2\*afstand

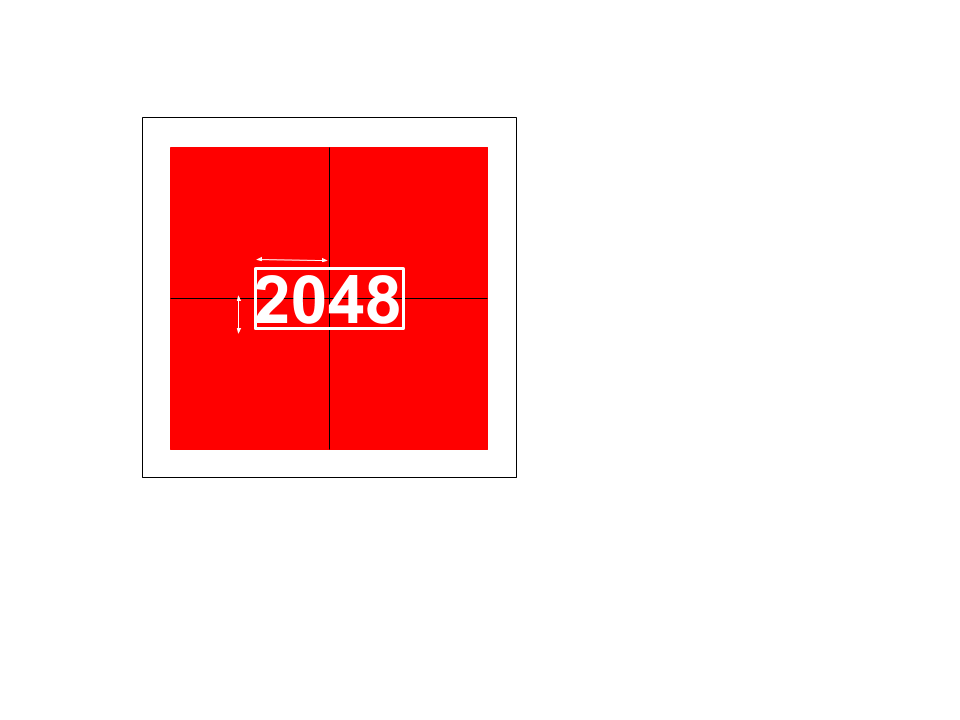
pygame.draw.rect(skærm, (255, 0, 0), (x, y, b, h))

tekst = font.render(str(bræt[rækkeI][klodsI]), True, (255, 255, 255))

skærm.blit(tekst, (x, y))

Prøv at kør det. Lav om på tallene oppe i brættet.

Hvordan sidder teksten? Ikke helt rigtig vel, nej. Det er fordi vi bare tegner det oppe hjørnet når vi egentlig gerne vil centrere det. Men hvordan gør man det?



Det gør man ved først at finde midtpunktet af selve klodsen og så trække halvdelen af tekstbilledets størrelse fra. Man finder selve klodsens midtpunkt ved at tage det øverste venstre hjørne og plusse med halvdelen af højden/bredden. Man finder størrelsen af tekstbilledet ved tekst.get\_size(), hvilket giver en liste tilbage med bredden og højden. Lad os nu beregne tekstens x- og y-koordinat i vores tegnKlods funktion.

def **tegnKlods**(rækkeI, klodsI):

x = klodsBredde \* klodsI + afstand

y = klodsHøjde \* rækkeI + afstand

b = klodsBredde - 2\*afstand

h = klodsHøjde - 2\*afstand

pygame.draw.rect(skærm, (255, 0, 0), (x, y, b, h))

tekst = font.render(str(bræt[rækkeI][klodsI]), True, (255, 255, 255))

tekstStørrelse = tekst.get\_size()

tekstX = x + b//2 - tekstStørrelse[0]//2

tekstY = y + h//2 - tekstStørrelse[1]//2

skærm.blit(tekst, (tekstX, tekstY))

Er det centreret? Prøv at ændre på tallene. Matematik magi er det eneste jeg har at sige!

## Step 2.4: Men hvad med farverne?

Noget af det ikoniske ved 2048 er dets farver. Så dem vil vi nu selv prøve at genskabe. Selve farverne finder vi på nettet, men måden vi gemmer dem, skal vi lige have på plads inden. Kan i huske vores dansk-engelsk-farveordbog? Fordi det er faktisk sådan en vi skal til at lave igen, men nu er det ikke fra et dansk ord til et engelsk, men fra et talklods tal til en farve i RGB-værdier. Hvor vi så slår op i vores dictionary når vi skal tegne vores firkant for at finde ud af hvilket farve den skal være. I mit eksempel nedenunder er 0 : rød, 2 : grøn, 4 : blå og 8 : gul.

farver = {0 : (255, 0, 0), 2 : (0, 255, 0), 4 : (0, 0, 255), 8 : (255, 255, 0)}

def **tegnKlods**(rækkeI, klodsI):

x = klodsBredde \* klodsI + afstand

y = klodsHøjde \* rækkeI + afstand

b = klodsBredde - 2\*afstand

h = klodsHøjde - 2\*afstand

pygame.draw.rect(skærm, farver[bræt[rækkeI][klodsI]], (x, y, b, h))

tekst = font.render(str(bræt[rækkeI][klodsI]), True, (255, 255, 255))

tekstStørrelse = tekst.get\_size()

tekstX = x + b//2 - tekstStørrelse[0]//2

tekstY = y + h//2 - tekstStørrelse[1]//2

skærm.blit(tekst, (tekstX, tekstY))

Prøv at ændre nogle af tallene i brættet til 2, 4 og 8, hvad ser i?

Nu skal vi så finde de rigtig farver så hop på nettet og find nogle billeder af 2048 og så find et pipetteværktøj frem så i kan finde RGB-værdierne og så skriv dem ind overfor det tal farven hører til. I kan bruge dette værktøj <https://imagecolorpicker.com/>.

Ligner det ikke 2048 nu?

## Step 3.1: Lille logik intermezzo!

Logikken følte jeg var lidt for svær at forklare her i ren tekst så den tog vi jo i fælleskab. Men nu hvor vi har lavet alle de funktioner vi skal bruge, skal vi få dem ind i vores andet program. Det er kun funktionerne vi skal bruge da vi allerede har et bræt. De funktioner vi skal have med ind i vores rigtige spil er som følger.

* roterVenstre
* roterHøjre
* bevægOp
* bevægNed
* bevægTilHøjre
* bevægTilVenstre
* rykTilVenstre
* tilføjTilfældig
  + Husk også i toppen af det rigtige program at ”import random” for det skal tilføjTilfældig-funktionen nemlig bruge.

Sørg for at få kopieret disse ind et sted efter vores farve dictionary og før vores while kør.

## Step 3.2: Hvordan var det nu man spillede?

Nu har vi koden til at rykke rundt på klodserne på pladen, men hvordan var det nu de kunne bevæge sig?

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Det her var hvad i selv konkluderede, hvilket jeg i den grad er meget enig i! Så vi skal altså lade spilleren rykke rundt på klodserne med pil-tasterne. Lad os starte med at gøre det.

Okay så vi har i vores spil allerede noget, der håndterer input fra spilleren. Men indtil videre er inputtet det kan håndtere begrænset til at trykke på det røde kryds op i hjørnet for at lukke programmet. Men det er det samme sted vi skal kunne håndtere andet input fra brugen nemlig trykken på pil-tasterne.

Pygame skelner mellem forskellige typer af input, hvor QUIT er en af typerne, men også KEYDOWN er en type input, hvilket sjovt nok er den type trykken på pil-tasterne har. Vi skal derfor lave en elif, hvor vi ser om inputtet er at der er blevet trykket på en tast. Hvis det er det skal vi så spørge om den tast henholdsvis er pilen til venstre (K\_LEFT) eller højre (K\_RIGHT) eller op (K\_UP) eller ned (K\_DOWN). Hvor vi for hver af disse tilfælde bevæger banen til den pågældende side og tilføjer en tilfældig klods (altså enten en to’er eller firer). Vores while kør: ender med at se således ud:

while kør:

tegn()

*#Vi løber igennem alle de ting der er sket. (input)*

for event in pygame.event.get():

if event.type == pygame.QUIT:

kør = False

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

bræt = bevægTilVenstre(bræt)

tilføjTilfældig()

elif event.key == pygame.K\_RIGHT:

bræt = bevægTilHøjre(bræt)

tilføjTilfældig()

elif event.key == pygame.K\_UP:

bræt = bevægOp(bræt)

tilføjTilfældig()

elif event.key == pygame.K\_DOWN:

bræt = bevægNed(bræt)

tilføjTilfældig()

Prøv nu dit spil er det ikke helt vildt meget ligesom 2048?????

## Step 3.3: Hvad mangler vi?

Så nu har vi jo hele spillet ikke? Lad os lige kigge på præsentationen en sidste gang for der er lige en ting vi stadig mangler.



Nemlig at man skal kunne tabe. Men hvornår er det man kan tabe? Er det bare når alle felterne er udfyldt? Nej, det er når at man ikke længere kan samle nogle af klodserne ved at rykke til nogen af siderne. Så hvis vi skal udtrykke det så en computer kan sige om det er rigtigt eller forkert kan vi jo bruge nogle af de funktioner vi har til bevæge klodserne til de forskellige sider. Men hvordan ved vi om den har bevæget nogle af klodserne? Det gør vi ved at vi kan sammenligne med brættet før og efter og se om de er det samme. Hvis de er det samme, har den ikke kunne rykke i den retning. Hvis det ikke ændrer brættet at rykke i alle fire retninger ved vi at spilleren har tabt. Så lad os prøve at lave en funktion, der kan lave denne vurdering for os og lad os kalde den brætUbevægelig. Vores bræt er en liste, hvilket er en mutable datatype, hvilket egentlig bare betyder at funktioner kan ændre på dens værdi så vi bliver nødt til at lave en kopi af den fordi vi ikke vil have at den fucker noget op. Så vi bruger et bibliotek, der hedder copy som vi skal huske at importere således ”import copy”. Lav funktionen inden while kør:

def **brætUbevægelig**(bræt):

brætVenstre = copy.deepcopy(bræt)

kanVenstre = bevægTilVenstre(brætVenstre) == bræt

brætHøjre = copy.deepcopy(bræt)

kanHøjre = bevægTilHøjre(brætHøjre) == bræt

brætOp = copy.deepcopy(bræt)

kanOp = bevægOp(brætOp) == bræt

brætNed = copy.deepcopy(bræt)

kanNed = bevægNed(brætNed) == bræt

return kanVenstre and kanHøjre and kanOp and kanNed

Nu skal vi se om den her funktion virker så lad os få at vide når vi ikke længere kan bevæge os ved at kalde den i vores game-loop sådan her:

while kør:

if brætUbevægelig(bræt):

**print**("NU KAN DU IKKE BEVÆGE DIG MERE")

tegn()

*#Vi løber igennem alle de ting der er sket. (input)*

for event in pygame.event.get():

if event.type == pygame.QUIT:

kør = False

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

.

.

.

Virker det, skriver den ud på det rigtige tidspunkt?

## Step 3.4: Nu skal spilleren lige få det at vide på en god måde ☺

Nu ved vi hvornår spilleren har ”tabt” og det skal vi også kunne fortælle dem. Nedunder laver jeg en slags ”slutskærm”, hvor jeg giver en bestemt besked, der indeholder den største klods de nåede at få. I kan prøve at lave jeres egen besked til spilleren eller bare bruge min.

def **tegnSlutskærm**():

*#Hvad er den højeste klods?*

størsteTal = 0

for række in bræt:

if **max**(række) > størsteTal:

størsteTal = **max**(række)

*#ÆSTETIK*

*#Først sort*

pygame.draw.rect(skærm, (0,0,0), (bredde//5, højde//5, bredde//5 \* 3, højde//5 \* 3))

*#Så en lidt mindre en med den farve som var den højeste spilleren opnåede*

pygame.draw.rect(skærm, farver[størsteTal], (bredde//5\*2, højde//5\*2, bredde//5, højde//5))

*#Så skriver vi lige en besked til slut :)*

tekst = font.render("Største klods var: "+str(størsteTal), True, (255, 255, 255))

tekstStørrelse = tekst.get\_size()

tekstX = bredde//2 - tekstStørrelse[0]//2

tekstY = højde//2 - tekstStørrelse[1]//2

skærm.blit(tekst, (tekstX, tekstY))

Nu skal vi lige sørge for at denne rent faktisk bliver tegnet, men også at den bliver tegnet på det rigtige tidspunkt. Så inden i vores tegn funktion skal vi kalde tegnSlutskærm når vi ved at man ikke kan bevæge brættet mere.

def **tegn**():

*#Gør hele skærmen sort*

skærm.fill((0,0,0))

*#Tegn bræt*

tegnBræt()

*#Tegn slutskærm*

if brætUbevægelig(bræt):

tegnSlutskærm()

*#Viser det vi har tegnet på skærmen*

pygame.display.flip()

Sådan får du noget op når man ikke længere kan bevæge den? Prøv at ændre beskeden eller nogle af de andre ting så spillet bliver lige som du vil have det!