# 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

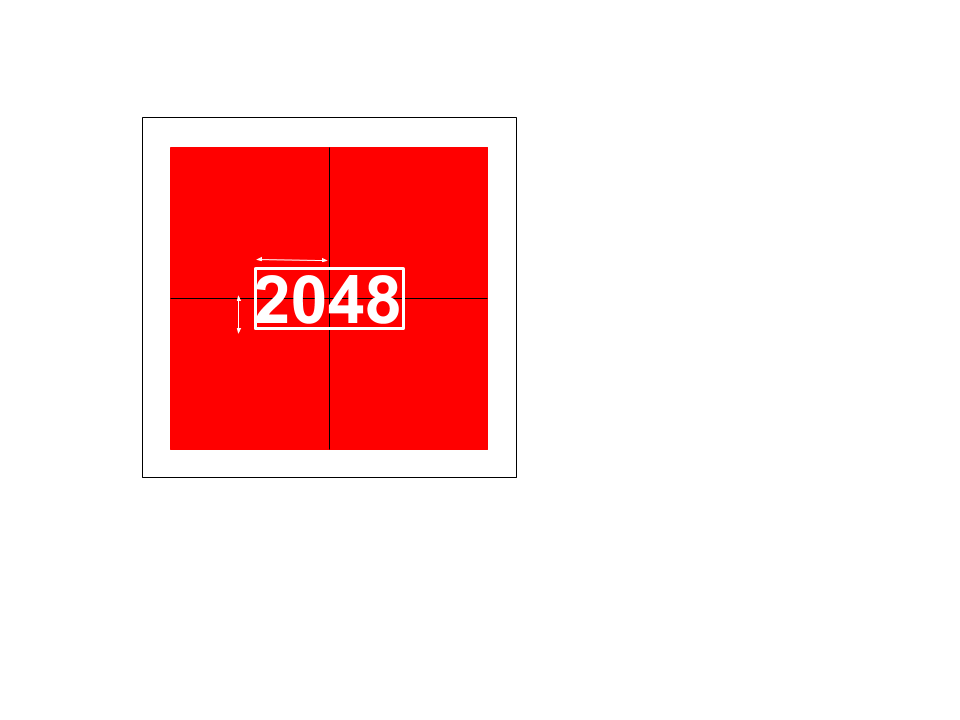
pygame.draw.rect(skærm, (255, 0, 0), (x, y, b, h))

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?