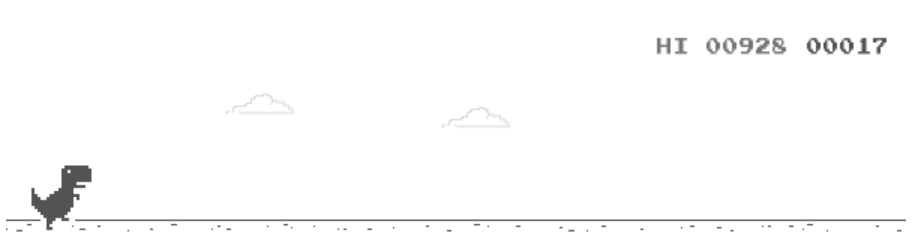
# Dino-spil

Når man starter på et nyt projekt, er det vigtigt at man altid overvejer hvad projektets bestanddele er så man kan komme ordentligt i gang. På billedet nedenunder ser vi dino spillet vi gerne vil abe efter.



Uhh, der er godt nok mange ting hva. Til at starte med vil vi fokusere på bare at får tegne dinoen og vores jord.

# Step 1: Tegn dinoen og jorden

Før vi overhovedet kan få tegnet dinoen og jorden på skærmen skal vi lige sætte nogle ting op i programmet først. Som i nok kan huske fra nogle af de andre spil, er der altid i toppen af filen nogle linjer hvor vi ”importerer” de ”biblioteker” (andre folks kode) vi skal bruge. Så det skal vi også gøre:

**import** **pygame**

Nu har vi fået pygame ind i vores program, men vi skal også lige bede pygame om at starte op.

*#Start pygame op*

pygame.init()

For at vi overhovedet kan tegne noget som helst skal vi lige lave det vindue vi kan tegne i, det gør vi ved at definere størrelses variablerne som vi også skal bruge senere (højde og bredde) og ved derefter at lave vores skærm variabel:

*#Højde og bredde af vores skærm*

højde = 300

bredde = 800

*#Lav skærmen med højde og bredde*

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

Nu skal vi lave variablerne der kommer til at indeholde størstedelen af den information spillet består af, nemlig de forskellige billeder, dinoens position og jordens position… Til at indeholde det laver vi en dictionary (ordbog) til både dinoen, jorden og spillet selv:

*#Spil variablen indeholder alle de værdier der er knyttet til spillets kørsel*

spil = {'kør':True}

*#Jord billedet og jord variablen der indeholder værdierne knyttet til jorden*

jord\_billede = pygame.image.load("ground.png")

jord = {'x':0, 'y':højde-19}

*#Jord billedet og jord variablen der indeholder værdierne knyttet til jorden*

dino\_billede = pygame.image.load("dino.png")

dino = {'x':120, 'y':højde - 95 - 10}

Grunden til at ’y’ værdierne i dinoens og jordens dictionary ser så spøjse ud er fordi vi jorden tilfælde vil have at bunden af jord billedet flugte med bunden af skærmen, så vi trækker jord højde i pixels fra højden af hele skærmen. I dinoens tilfælde vil vi gerne have at dinoens fødder flugter med toppen af jorden, så vi trækker både højden af dino billedet og højden af jord billedet fra hele højden af skærmen.

Nu er vi klar til at lave funktionen som skal tegne det hele. Den funktion kalder vi tegn. Ligesom i det rigtige dinospil vil vi også gerne have en hvid baggrund, så skal vi tegne vores dino og jord, hvor vi andvender de koordinater vi har defineret i dino og jord dictionariesne, sidst men ikke mindst sørger vi for at det hele bliver vist på skærmen:

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x'], jord['y']))

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

Nu har vi alle de nødvendige variabler og funktioner til at tegne dinoen og jorden. Men for at vores spil ikke bare bliver åbnet og kører for evigt skal vi give spilleren en mulighed for at lukke programmet ned. Måden vi gør det på er samme måde som vi ville opfange andet spiller input. Men der er også andre situationer hvor spillet skal lukke ned f.eks. når dinoen dør. Derfor laver vi det såkaldte spilløkke, i spil løkken har vi alle de ting vores spil skal blive ved med at gøre indtil spillet skal lukkes og spilløkken stoppes. Til det bruger vi værdien i vores spil variabel der hedder kør. Sådan ser vores spilløkke ud. Prøv at læs den, kan i forstå hvad hver linje gør?

*#Spil loop*

**while** spil['kør']:

*#For hver begivenhed(tastatur tryk eller ligende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

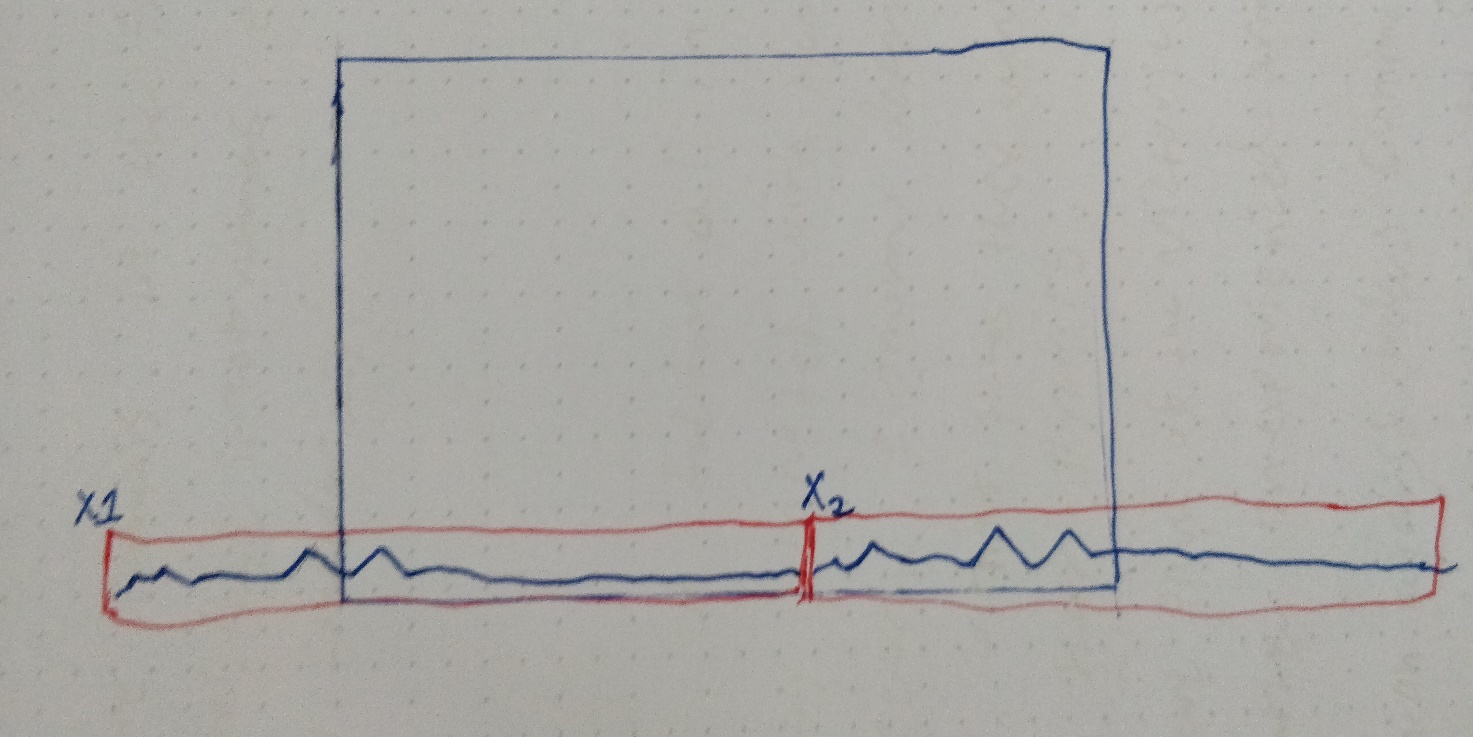
*#Tegn spillet*

tegn()

Kør spillet, kommer der noget op?

## Step 2: Få jorden til at kører

Som i nok kan huske fra spillet er det ikke dinoen der bevæger sig (i hvert fald ikke vandret), men derimod jorden der kører til venstre. Det skal vi også have så det ikke ser helt så kedeligt ud. Måden vi for det til at ske på er ved rent faktisk at ændre position af jordbilledet, men hvis bare bliver ved med at rykke billedet til venstre vil der på et tidspunkt ikke være mere billede tilbage. En løsning på det er ved at tage og tegne to billeder hver gang, men med hver deres x-position:



Som vi kan se på billedet oven over tegner vi jordbilledet to gange, så når x2 er lig 0 (og det første billede er helt ude af skærmen) starter de forfra.

For at vores spil skal kunne dette skal vi have lidt mere kode på. Ligesom på illustrationen skal vi have to x koordinater et til hvert billede, derudover skal vi senere hen bruge bredden af vores jordbilledet, vi ændrer derfor jord variablen til det følgende:

jord = {'x1':0, 'x2':1203 - 0, 'y':højde-19, 'bredde':1203}

Nu skal vi lave den funktion, der for jorden til at rykke sig, den kalder vi jord\_opdater. I den skal vi starte med at rykke x1 tilbage, derefter sørger vi for at x2 beholder sin position lige efter x1, ved at tage bredden af jordbilledet og plusse med den negative x1. Derefter skal vi tjekke om det første billede er røget helt uden for, hvis ja sætter vi x1 til at være lig 0:

**def** jord\_opdater():

*#Ryk x1*

jord['x1'] -= 20

*#Sørg for at x2 forbliver lige ved slutningen af billede 1*

jord['x2'] = jord['bredde'] + jord['x1']

*#Er x1 større end eller lig med bredden af jorden?*

*#Grunden til vi ganger med -1 er fordi x1 vil være negativ og vi dermed laver det om til et positivt tal*

**if** jord['x1'] \* -1 >= jord['bredde']:

*#Hvis ja, sæt x1 lig med 0*

jord['x1'] = 0

Så er vi klar til at tegne de to billeder. For at gøre det skal vi ændrer vores tegn funktion, således jord billedet bliver tegnet to gange, henholdsvis med x1 og x2. Hvor vi til sidst får programmet til at vente i et kort stykke tid (30 milisekunder) ved hjælp af funktion pygame.time.delay(30), så det ikke kører for hurtigt. Tegn funktionen bliver dermed:

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

*#Vent 30 milisekunder*

pygame.time.delay(30)

Nu skal vi bare sørge for at vi opdaterer jorden, det gør vi ved at kalde jord\_opdater funktionen det samme sted som vi kalder vores tegn funktion, nemlig i spilløkken. Den kommer dermed til at se sådan ud:

**while** spil['kør']:

*#For hver begivenhed(tastatur tryk eller ligende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Opdater jordens position*

jord\_opdater()

*#Tegn spillet*

tegn()

Bevæger jorden sig? Bevæger den sig for hurtigt for langsomt? Hvor kan du ændre i det?

## Step 3: Få dinoen til at løbe

Som I nok kan se, ser dinoen lidt mærkelig ud. Grunden til den gør det er fordi dinoen ikke løber! Så det vi skal nu, er nemlig at få den til at løbe. Som I måske har lagt mærke til var der en masse billeder i folderen I downloadede. En af dem er et billede, der hedder dinoer.png. På dinoer.png kan i se der er mange forskellige versioner af næsten den samme dino, hvor det tredje og fjerde er dinoen med henholdsvis det ene og det andet ben løftet. Det er dem vi gerne vil skifte i mellem. For at vi ved hvilket billede af dinoen det er vi skal tegne skal vi tilføje endnu et element til dinoens dictionary. Derudover skal vi også vide hvor bredt et af billederne er, hvor højt det er og så skal vi sætte dino\_billede til billedet med alle dinoerne. Dermed bliver dinoens dictionary og dino\_billede til:

dino\_billede = pygame.image.load("dinoer.png")

dino = {'x':120, 'y':højde - 95 - 10, 'bredde':440//5, 'højde':95,'billed\_nr':2}

Nu skal vi også sørge for at vores dino rent faktisk skifter sit billede, så ligesom vi lavede en funktion, der stod for at opdatere vores jord, vil vi også have en funktion, der opdaterer vores dino. Den kan vi kalde dino\_opdater, siden det kun er billede 2 og 3 vi vil se, vil vi ændre billed\_nr til 3 hvis den var 2 og til 2 hvis den var 3. Dino\_opdater kommer til at se sådan her ud:

**def** dino\_opdater():

**if** dino['billed\_nr'] >= 3:

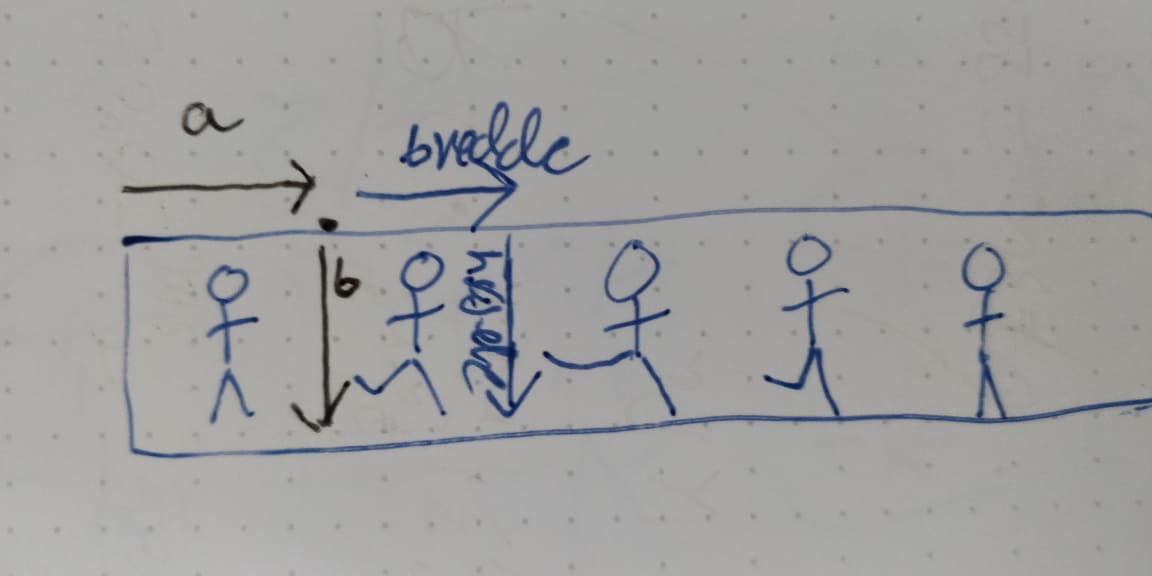
dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 1

Vi vil kalde dino\_opdater funktion lige under hvor vores jord\_opdater funktion bliver kaldt.

Nu skal vi sørge for at det kun er nået af dinoen, der bliver tegnet. Pygames skærm.blit(…) funktion har en måde hvor vi kun viser en del af billedet, ved at man giver den en tuple(en liste man ikke kan lave om på angivet med (parenteser) ). Det man skal give i den tuple er hvor mange pixels inde og nede i billedet man vil begynde at vise fra og så hvor mange pixels til højre og ned af man derfra vil vise: (a, b , bredde , højde)



Vores tegn funktion kommer dermed til at se sådan ud:

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']), (dino['bredde']\*dino['billed\_nr'], 0, dino['bredde'], dino['højde']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

*#Vent 30 milisekunder*

pygame.time.delay(30)

Kør det bevæger benene sig? Hvor hurtigt, ligner det at den rigtigt løber?

## Step 4: Få dinoen til ikke at løbe så hurtigt

Som i nok kunne se gik det lidt stærkt for den dino. Lad os få det til at gå lidt langsommere. Måden vi kan gøre det på er at den ikke skal skifte billede lige så ofte, så vi skal dermed ændre det sted vi får den til at skifte billede og der er i dino\_opdater funktionen. I stedet for at vi plusser med en hver gang den bliver kaldt så lad os i stedet plusse med lidt mindre. For at sørge for at begge billeder bliver vist i cirka lige lang tid skal vi også ændre hvornår vi ændrer billedet tilbage til billede 2. dino\_opdater funktionen kommer til at se sådan ud:

**def** dino\_opdater():

**if** dino['billed\_nr'] >= 3.5:

dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 0.4

Men vi vil ikke have at man bare glider fra billede til billede, derfor skal vi sørge for at vi kun ganger med hele tal, det kan int() funktion hjælpe os med. Den laver et tal til et heltal (ved at runde op eller ned). Så vores tegne funktion kommer til at se sådan ud:

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']), (dino['bredde']\*int(dino['billed\_nr']), 0, dino['bredde'], dino['højde']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

*#Vent 30 milisekunder*

pygame.time.delay(30)

Hvordan ser det ud? Er det ik lidt bedre : )

## Step 5: Nu skal der hoppes

Hvordan søren får vi dinoen til at hoppe? Hvad er at hoppe? Det er jo ikke lige så let som at gå til venstre og højre, fordi han skal ryge op i luften og derefter ryge ned igen. Men ved i hvad nu kan en af verdenens ældste videnskabelige discipliner kommer os til undsætning. Den videnskab er fysikken, den har en formel, der kan hjælpe os med at få vores dino til at hoppe den ser sådan her ud:

Det den formel viser er hvor meget kraft der udøves på et objekt hvor F er kraften, m er massen, v er hastigheden. Hvis vores dino hopper afsætter dens ben den med en bestemt hastighed, lad os bare sige 8 og vores dino vejer 2. Men imens dinoen hopper trækker tyngdekraften stadig ned på ham og hans hastighed formindskes stille og roligt. På et tidspunkt er dinoens hastighed lavere end nul således dinoen begynder at bevæge sig nedad, det vil han gøre hurtigere og hurtigere indtil han når jorden. Nu skal vi kode den her opførsel!!!!

Til at starte med skal vi lige give dinoens dictionary nogle flere værdier, nemlig: en om han hopper eller ej, en med hans hastighed, en med hans masse og en der siger nu kan du ikke komme længere ned. Dinoens dictionary kommer til at se sådan her ud:

dino = {'x':120, 'y':højde - 95 - 10, 'bredde':440/5, 'højde':95, 'billed\_nr':2, 'hopper':False, 'v':8, 'm':2, 'y\_min': højde - 95 - 10}

Nu skal vi lave den funktion, der skal sørge for at dinoen rent faktisk hopper når din[’hopper’] er sand (True). Den funktion kalder vi dino\_hopper. I den skal vi udregne hvor meget vi skaal ændre på dinoens y position(F) med formlen ovenover. Derudover skal vi ændre på farten vores dino bevæger sig med op ad. Til sidst men ikke mindst skal vi tjekke om dinoen har ramt jorden igen, hvis det er tilfældet så skal sige dinoen ikke hopper mere og sætte farten til den gamle ”position”. Funktionen dino\_hopper kommer til at se således ud:

**def** dino\_hopper():

*#Vi tjekker om v er større end nul, hvis det er tilfældet, vil vi gerne gøre y-værdien mindre (bevægelse op ad)*

**if** dino['v'] > 0:

F = ((dino['m'] \* (dino['v']\*dino['v']))/2)

*#hvis v er mindre end nul, vil vi gøre y-værdien større (bevægelse nedad), ved at gøre F negativ*

**else**:

F = -((dino['m'] \* (dino['v']\*dino['v']))/2)

*#Vi rykker dinoen op eller ned med vores før udregnede F*

dino['y'] -= F

*#Vi sænker dinoens hastighed*

dino['v'] -= 1

*#Til sidst tjekker vi om dinoen har nået jorden igen*

**if** dino['y'] >= dino['y\_min']:

dino['y'] = dino['y\_min']

dino['hopper'] = False

dino['v'] = 8

Nu har vi lavet de underliggende fuktioner således vores dino KAN hoppe. Men vi kalder dem ingen steder og vi tjekker aldrig om, der er blevet trykket på den rigtige knap (pil-op).

Lad os starte med at tjekke om spilleren vil have dinoen til at hoppe, det gør vi ved at tjekke om pil-op tasten er trykket ned. Hvis den er trykket ned skal vi sætte dino[’hopper’] til True. Det kan vi gøre samme sted i game loopet som hvor vi tjekker om escape-tasten er blevet trykket ned. Game loopet kommer til at se således ud:

**while** spil['kør']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hopper'] = True

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Tegn spillet*

tegn()

Nu skal vi som det allersidste sørge for at dino\_hopper funktionen rent faktisk bliver kaldt. Hvornår skal dino\_hopper funktionen kaldes? Den skal selvfølgelig kaldes når dinoen hopper (når dino[’hopper’] er True). Det kan vi tjekke om den er i vores dino\_opdater funktion, koden til den funktion kommer til at se sådan her ud:

**def** dino\_opdater():

*#Hopper dinoen*

**if** dino['hopper']:

dino\_hopper()

*#Skift dinoens animation*

**if** dino['billed\_nr'] >= 3.5:

dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 0.4

Prøv at se hopper han, hvordan hopper han? Hopper han højt nok, evt. for højt? Hvilke værdier kan du lave om for at ændre på det?

## Step 6: Kan du høre det?

Nu kan vores dino hoppe, men det er som om der mangler et eller andet. Nemlig lyden! Hop ind på det her link( [www.kortlink.dk/xffv](http://www.kortlink.dk/xffv)) og download lyden hop.wav, derefter skal du lægge den i dino mappen. Nu skal vi prøve at få denne lyd til at spille når dinoen hopper. Til at starte med skal vi lige få lyden ind i vores program, der bruger vi funktionen pygame.mixer.Sound(fil), tilføj en værdi til dino dictionarien med nøglen ”hop lyd”. Så dino dictionarien kommer til at se sådan her ud:

dino = {'x':120, 'y':højde - 95 - 10, 'bredde':440//5, 'højde':95, 'billed\_nr':2, 'hopper':False, 'v':8, 'm':2, 'y\_min': højde - 95 - 10, 'hop lyd':pygame.mixer.Sound("hop.wav")}

Nu har vi fået lyden ind, nu skal vi også spille den. Det skal vi gøre der hvor vi tjekker om dinoen skal hoppe eller ej. Så vores gameloop ender med at se sådan her ud:

**while** spil['kør']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hop lyd'].play()

dino['hopper'] = True

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Tegn spillet*

tegn()

Kan du høre det?

## Step 7: Dino vs. ?

Nu kan vores dino hoppe og alt muligt, men den mangler lidt noget at hoppe over, ik?

Lad os starte med at få kaktussen på skærmen. For at få den på skal vi ligesom med jorden og dinoen, loade dens billede og lave en dictionary tilhørende kaktussen. Kaktus billedet har flere forskellige kaktusser, så vores dictionary skal udover dens position også indeholde både bredde, højde og billed\_nr, men det er ligegyldigt hvilken kaktus vi ser, men de må gerne være forskellige. Koden der indeholder det, ser sådan her ud:

*#Kaktus billedet og kaktus variablen der indeholder værdierne knyttet til kaktussen*

kaktus\_billede = pygame.image.load("kaktus.png")

kaktus = {'x':bredde, 'y':højde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5)}

Som i nok kan se er kaktussens billed\_nr lidt anderledes, den bruger nemlig funktionen randint, der tager et tilfældigt heltal, i dette tilfælde mellem 0 og 5. Men før vi kan bruge den skal vi importere random biblioteket. Så allerøverst skal du tilføje denne linje:

**import** **random**

Nu skal vi få kaktussen til at rykke sig, derudover skal vi finde ud af om den har ramt venstre side af skærmen, hvis ja skal kaktussen ”starte forfra”, vi vil dog ikke have at den kommer præcist på samme tid hver gang, derfor tilføjer vi en tilfældig værdi. Det vil vi gøre i vores kaktus\_opdater funktion, som kommer til at se sådan her ud:

**def** kaktus\_opdater():

*#Flyt kaktussen*

kaktus['x'] -= 20

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** kaktus['x'] <= 0:

*#Kaktussen bliver sat tilbage + en tilfældig værdi mellem 0 og 200 så den ikke kommer helt på samme tidspunkt hele tiden*

kaktus['x'] = bredde + random.randint(0, 200)

kaktus['billed\_nr'] = random.randint(0, 5)

Du skal kalde kaktus\_opdater der hvor du kalder de andre opdater funktioner.

Nu mangler vi bare at tegne kaktussen, vi skal tegne den lidt på samme måde som med vores din, hvor vi tager højde for hvilket billede af kaktussen det er der skal tegnes. Tegn funktionen kommer til at se sådan her ud:

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']), (dino['bredde']\*int(dino['billed\_nr']), 0, dino['bredde'], dino['højde']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Tegn kaktussen*

skærm.blit(kaktus\_billede, (kaktus['x'], kaktus['y']), (kaktus['b']\*kaktus['billed\_nr'], 0, kaktus['b'], kaktus['h']))

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

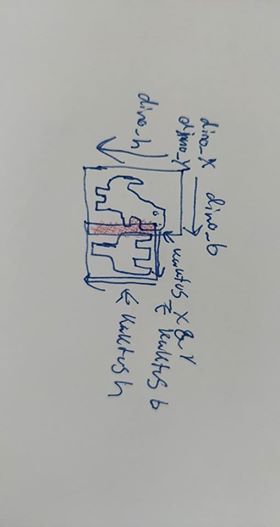
*#Vent 30 milisekunder*

pygame.time.delay(30)

Prøv at kør det, hvordan er kaktussen?

## Step 8: Kaktusser gør nas

Vores spil er en lille smule urealistisk, fordi hvad sker hvis man rammer en kaktus? Ja der sker det at det gør ondt på en, men vores dino opdager det slet ikke! Det skal vi ændre på. For at vi kan det skal vi lige have fundet en måde hvor vi kan se om dinoen og kaktussen rammer hinanden, men hvordan gør man det?



Lad os tage dette eksempel, vi ved at der hvor det er markeret med rødt, der ”overlapper” de to figurer hinanden, men hvordan kan vi vide det. Det vi fordi at dino\_x + dino\_b > kaktus\_x, det vil sige at dinoen ihvertfal befinder sig tildels på venstre side af kaktussen, men for at vide om den er helt udenfor eller ej bliver nødt til at vide om dino\_x < kaktus\_x + kaktus\_h, hvis begge er sande ved vi at de på x-aksen er indenfor hinanden, vi gør det samme for y aksen.

For at få det ind i vores kode laver vi en funktion der hedder dino\_ramt, hvor vi vil stoppe spillet hvis dinoen bliver ramt. Men vi vil gerne kunne teste om dinoen er blevet ramt af hvad som helst, derfor tager vi det der før var kaktussens koordinater osv. som argumenter til funktionen. Den kommer dermed til at se sådan her ud:

**def** dino\_ramt(x, y, b, h):

**if** dino['x']+dino['bredde'] > x **and** dino['x'] < x+b **and** dino['y']+dino['højde'] > y **and** dino['y'] < y+h:

spil['kør'] = False

Du kan nu kalde dino\_ramt i din dino\_opdater funktion, hvor du giver den kaktussens x, y, bredde og højde.

Så dino\_opdater funktionen kommer til at se sådan her ud:

**def** dino\_opdater():

*#Har dinoen ramt kaktussen*

dino\_ramt(kaktus['x'], kaktus['y'], kaktus['bredde'], kaktus['højde'])

*#Hopper dinoen*

**if** dino['hopper']:

dino\_hopper()

*#Skift dinoens animation*

**if** dino['billed\_nr'] >= 3.5:

dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 0.4

Prøv det, hvad sker der når dinoen rammer kaktussen?

## Step 9: Score

Nu hvor vores kaktusser er blevet farlige skal vi også have en måde at se hvor mange vi rent faktisk har klaret. Vi har brug for en score. Vi vil gerne give spilleren 1 point hver gang han har hoppet over en kaktus, vi skal derfor tilføje to værdier, en til vores spil dictionary og en til vores kaktus dictionary. De to værdier skal henholdsvis være scoren og om kaktussen er blevet hoppet over. Vores spil og kaktus dictionaries kommer dermed til at se således ud:

*#Spil variablen indeholder alle de værdier der er knyttet til spillets kørsel*

spil = {'kør':True, 'score':0}

og

kaktus = {'x':bredde, 'y':hjde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5), 'hoppet over':False}

Det er bedst at give et point så hurtigt så muligt, så vi vil give et point til spilleren, hvis han har passeret kaktussen og han ikke allerede har hoppet over den. Men vi skal derfor også huske at ændre kaktussens hoppet over status efter første gang så vi ikke får for mange point. Derudover skal vi huske at ”resette” vores hoppet over værdi når kaktussen sættes tilbage til den anden side. Vores kaktus\_opdater funktion kommer til at se sådan her ud:

**def** kaktus\_opdater():

*#Flyt kaktussen*

kaktus['x'] -= 20

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** kaktus['x'] <= 0:

kaktus['x'] = bredde + random.randint(0, 200)

kaktus['billed\_nr'] = random.randint(0, 5)

kaktus['hoppet over'] = False

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > kaktus['x']+kaktus['bredde'] **and** **not** kaktus['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

kaktus['hoppet over'] = True

Nu har du implementeret et point system, prøv det.

## Step 10: Har man virkelig point, hvis man ikke kan se dem?

Det er lidt frustrerende at man ikke kan se det antal point man har fået, det skal vi lave om på. Vi laver derfor en funktion, der hedder visScore. Den renderer teksten ligesom I (forhåbentligt) kan huske fra nogle af de andre spil. Den starter med at hente en font, derefter renderer den teksten vi vil vise på skærmen og sidst, men ikke mindst, viser den det på skærmen. Koden kommer dermed til at se sådan her ud.

**def** visScore():

font = pygame.font.SysFont("Arial", 20)

*#Rendér scoren, ved først at lave det til en streng*

score\_tekst = font.render('Score: '+str(spil['score']), True, (0,0,0))

*#Prop de teksten på skærmen, get\_rect() funktionen giver en liste med fire tal tilbage, [x, y, bredde, højde], vi minusser med breden så teksten kommer til at stå yderst til venstre*

skærm.blit(score\_tekst, (bredde - score\_tekst.get\_rect()[2], 5))

Du skal nu kalde visScore funktionen i din tegn funktion. Var det ikke bedre?

## Step 11: En score er fin nok, men det er high scoren der virkelig gælder!

Det rigtige spil har en high score, det er den eneste man kigger på ikke, det er den man vil slå. Så nu skal vi også have en high score. Det laver vi ved at tilføje endnu en værdi til spil dictionarien og så skal vi også tegne den værdi.

Vi starter med at tilføje high score værdien til vores spil dictionary, ændre den så den ser sådan her ud:

spil = {'kør':True, 'score':0, 'high score':0}

Nu skal vi sørge for at high score værdien også bliver tegnet, vi vil gerne tegne den i samme ”sætning” som vi tegner vores nuværende score. Så visScore funktionen kommer til at se sådan her ud:

**def** visScore():

font = pygame.font.SysFont("Arial", 20)

*#Rendér scoren, ved først at lave det til en streng*

score\_tekst = font.render('High Score: '+str(spil['high score'])+' '+'Score: '+str(spil['score']), True, (0,0,0))

*#Prop de teksten på skærmen, get\_rect() funktionen giver en liste med fire tal tilbage, [x, y, bredde, højde], vi minusser med breden så teksten kommer til at stå yderst til venstre*

skærm.blit(score\_tekst, (bredde - score\_tekst.get\_rect()[2], 5))

Sidst men ikke mindst skal vi sørge for at vi rent faktisk sørge for at sætte high scoren til scoren fra spillet, hvis scoren i spiller er højere end den nuværende high score. Vi tilføjer derfor følgende kode helt under vores game loop:

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

spil['score'] = 0

Virker det? Kan du se din high score?

## Step 12: Er det sådan en high score fungerer?

Som i nok lagde mærke til er en high scorer ikke så meget værd hvis man ikke kan spille mere end en gang, så man aldrig kan slå den! Vi skal derfor sørge for at vi kan spille mere end en gang. Det gør vi ved at smide vores nuværende loop ind i et andet loop. Men før vi kan det skal vi lige revurdere, fordi vores dino skal stadig kunne dø, så vi skal også have tilføjet en værdi, der fortæller os om dinoen er død i stedet for at vi anvender værdien, der fortæller os om spillet skal kører i det inderste loop. Udover det skal vi også have lavet en game over skærm, hvor vi kan se om brugeren vil spille igen.

Vi starter med at tilføje en ”levende” værdi til vores dino dictionary, den skal vi bruge i det inderste loop:

dino = {'x':120, 'y':hojde - 95 - 10, 'bredde':440//5, 'højde':95, 'billed\_nr':2, 'hopper':False, 'v':8, 'm':2, 'y\_min': hojde - 95 - 10, 'hop lyd':pygame.mixer.Sound("hop.wav"), 'levende':True}

Nu skal vi også lige ændr der i dino\_ramt funktionen, hvor vi vil vide om dinoen er død, i stedet for at sætte spil kør værdien til false vil vi sætte dinoens levende værdi til false:

**def** dino\_ramt(x, y, b, h):

**if** dino['x']+dino['bredde'] > x **and** dino['x'] < x+b **and** dino['y']+dino['højde'] > y **and** dino['y'] < y+h:

dino['levende'] = False

Nu vil vi skrive vores gameover funktion, der tegner noget tekst på skærmen, der informerer$ brugeren om deres død, den ligner lidt vores visscore og kommer til at se sådan her ud:

**def** gameOver():

font = pygame.font.SysFont("Arial", 50)

*#Renderer teksten "Game Over!"*

tekst = font.render("Game Over", True, (0,0,0))

*#Prop de teksten på skærmen*

skærm.blit(tekst, (bredde/2 - tekst.get\_rect()[2]/2, højde/2 - tekst.get\_rect()[3]/2 ))

pygame.display.flip()

Nu har vi lavet de nødvendige funktioner, nu kan vi lave det nye loop unden om gameloopet. I det nye loop skal vi tjekke om scoren er større end highscoren som vi kodede sidste gang, så skal vi se om der bliver trykket på hvilken som helst knap, så vi kan starte spillet igen. Sidst men ikke mindst skal vi også kalde vores gameover funktion. Den kommer til at se sådan her ud (ps. Bare ryk det tidligere gameloop ind og ændr’ ):

**while** spil['kør']:

gameOver()

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis hvilken som helst anden tast trykkes ned, så genstart spillet og sæt kaktussen tilbage*

**else**:

dino['levende'] = True

dino['hopper'] = False

dino['y'] =

kaktus = {'x':bredde, 'y':højde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5), 'hoppet over':False}

**while** dino['levende']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hopper'] = True

dino['hop lyd'].play()

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Opdater kaktussen*

kaktus\_opdater()

*#Tegn spillet*

tegn()

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

spil['score'] = 0

## Step 13: Nu går det lidt langsomt

Jaer, nu hvor vi har fået en high score på, lægger man virkelig mærke til hvor langsomt det går. Vi gerne have det til at gå hurtigere jo længere tid man har overlevet. Det kan vi gøre ved at tilføje en fart værdi til vores spil dictionary og så få jorden og kaktussen til at bevæge sig relativt med den. Denne fart værdi skal selvfølgelig blive en lille smule større hele tiden. Det gør vi således.

Vi starter med at give vores spil dictionary endnu en værdi, nemlig fart værdien:

spil = {'kør':True, 'score':0, 'high score':0, 'fart':1}

Herefter skal vi sørge for at jorden og kaktussen bevæger sig relativt med fart værdien, det gør vi ved at gange hvor meget de bevæger sig med hele tiden nu med fart værdien så de kommer til at se henholdsvis sådan her ud:

**def** jord\_opdater():

*#Ryk x1*

jord['x1'] -= 20\*spil['fart']

*#Sørg for at x2 forbliver lige ved slutningen af billede 1*

jord['x2'] = jord['bredde'] + jord['x1']

*#Er x1 større end eller lig med bredden af jorden?*

*#Grunden til vi ganger med -1 er fordi x1 vil være negativ og vi dermed laver det om til et positivt tal*

**if** jord['x1'] \* -1 >= jord['bredde']:

*#Hvis ja, sæt x1 lig med 0*

jord['x1'] = 0

og sådan her:

**def** kaktus\_opdater():

*#Flyt kaktussen*

kaktus['x'] -= 20\*spil['fart']

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** kaktus['x'] <= 0:

kaktus['x'] = bredde + random.randint(0, 200)

kaktus['billed\_nr'] = random.randint(0, 5)

kaktus['hoppet over'] = False

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > kaktus['x']+kaktus['bredde'] **and** **not** kaktus['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

kaktus['hoppet over'] = True

Nu skal vi bare resette fart værdien, hvis spillet skal starte forfra. Når vi er inde i det inderste loop skal vi hele tiden sætte fart værdien en smule op. De to loops kommer til at se sådan her ud:

**while** spil['kør']:

gameOver()

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis hvilken som helst anden tast trykkes ned, så genstart spillet og sæt kaktussen tilbage*

**else**:

dino['levende'] = True

dino['hopper'] = False

dino['y'] = højde-70-10

kaktus = {'x':bredde, 'y':højde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5), 'hoppet over':False}

spil['fart'] = 1

**while** dino['levende']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hopper'] = True

dino['hop lyd'].play()

*#Opdater farten*

spil['fart'] += 0.0003

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Opdater kaktussen*

kaktus\_opdater()

*#Tegn spillet*

tegn()

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

spil['score'] = 0

Går det for hurtigt eller ikke hurtigt nok, så ændr’ hvor meget du sætter farten op med.

## Step 14: Var der ikke også noget andet end kaktusser?

I det rigtige spil var der vist noget andet end kaktusser, nemlig fugle. Hvis i kigger i mappen kan i også se at der er et fugle billede. Så lad os få den ind i vores spil. Vi starter ligesom med dinoen og kaktussen med at loade billedet ind og lave en dictionary med de nødvendige værdier, den kommer til at se sådan her ud:

*#Fugle billedet og fugle variablen der indeholder værdierne knyttet til fuglen*

fugl\_billede = pygame.image.load("fugl.png")

fugl = {'x':bredde, 'y':højde-81-10, 'bredde':94, 'højde':81, 'billed\_nr':0, 'hoppet over':False}

Hvis i kan huske fra spillet basker fuglen med vingerne, billedet af fuglen fra mappen har også to slags vingeslag, vi skal derfor ligesom med dinoen skifte mellem de her to billeder, derfor har vi at billed nr. er 0 og så skal vi også ændre det i vores fugl\_opdater funktion. Vi gør ligesom med dinoen for ikke at få den til at flyve alt for hurtigt. Udover at ændre billedet skal den gøre meget af det samme som kaktus\_opdater funktionen.

**def** fugl\_opdater():

*#Flyt kaktussen*

fugl['x'] -= 20\*spil['fart']

*#Skift fuglens billede*

**if** fugl['billed\_nr'] >= 1.5:

fugl['billed\_nr'] = 0

**else**:

fugl['billed\_nr'] += 0.4

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** fugl['x'] <= 0:

fugl['x'] = bredde + random.randint(0, 200)

fugl['billed\_nr'] = 0

fugl['hoppet over'] = False

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > fugl['x']+fugl['bredde'] **and** **not** fugl['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

fugl['hoppet over'] = True

Først skal vi sørge for at kalde fugl\_opdater funktionen nede ved vores andre funktioner. Derudover skal vi sørge for at ”resette” vores fugl dictionary, som vi gør det med kaktus dictionarien, i det yderste gameloop. (Spørg hvis i ikke helt forstår) Vi skal også kunne dø af vores fugl, derfor skal vi tilføje en linje i dino\_opdater funktionen under der hvor vi tjekker om man har ramt kaktussen. Den ender med at se sådan her ud:

**def** dino\_opdater():

*#Har dinoen ramt kaktussen*

dino\_ramt(kaktus['x'], kaktus['y'], kaktus['bredde'], kaktus['højde'])

*#Har dinoen ramt fuglen*

dino\_ramt(fugl['x'], fugl['y'], fugl['bredde'], fugl['højde'])

*#Hopper dinoen*

**if** dino['hopper']:

dino\_hopper()

*#Skift dinoens animation*

**if** dino['billed\_nr'] >= 3.5:

dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 0.4

Nu skal vi også lige huske at tegne den, vi gør ligesom med dinoen, da vi ved at vores billed\_nr ikke nødvendigvis er et helt tal. Så vores tegn funktion kommer til at se sådan her ud.

**def** tegn():

*#Tegn hvid baggrund*

skærm.fill((255,255,255))

*#Tegn dinoen*

skærm.blit(dino\_billede, (dino['x'], dino['y']), (dino['bredde']\*int(dino['billed\_nr']), 0, dino['bredde'], dino['højde']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Tegn kaktussen*

skærm.blit(kaktus\_billede, (kaktus['x'], kaktus['y']), (kaktus['bredde']\*kaktus['billed\_nr'], 0, kaktus['bredde'], kaktus['højde']))

*#Tegn fuglen*

skærm.blit(fugl\_billede, (fugl['x'], fugl['y']), (fugl['bredde']\*int(fugl['billed\_nr']), 0, fugl['bredde'], fugl['højde']))

*#Tegn scoren*

visScore()

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

*#Vent 30 milisekunder*

pygame.time.delay(30)

Prøv at kør det, i hvilken rækkefølge kommer de?

## Step 15: Helst ik på samme tid

Hvad de kører OVEN I HINANDEN, ej det går altså ikke. For at det kun er en fjende vi kan se ad gangen, så lad os tilføje en værdi til vores spil dictionary som vi kalder for fjende. Nu skal vi dog passe på fordi vores spil dictionary ligger over vores to fjende dictionaries (fugl og kaktus). Vi kan derfor først indsætte fjende værdien efter vores fugl dictionary, vi skriver derfor følgende under fugl dictionarien.

*#Vi tilføjer fjende værdien til spil dictionarien*

spil['fjende'] = [kaktus, fugl][random.randint(0,1)]

En lille forklaring af koden ovenover, grunden til at vi kan tilføje en ny værdi på samme måde som vi har ændret de andre er fordi vi laver et nyt nøgleord ”fjende”, som python ikke finder i spil dictionarien, den tilføjer det bare med den værdi der står på den anden side af lighedstegnet. En lille forklaring af det, der står på højre side af lighedstegnet: vi laver en liste med kaktussen og fuglen og vælger en tilfældig en af dem ved at vælge tilfældigt mellem indeks 0 eller 1.

Nu skal vi sørge for at vi kun opdatere den valgte fjende. For at vide hvilken en der skal opdateres kan vi bare tilføje den pågældendes fjendes (enten kaktussens eller fuglens) opdater funktion til dens egen dictionary. Det gør vi ligesom da vi tilføjede fjende værdien til vores spil dictionary efter kaktus og fugl dictionarierne. Så efter kaktus\_opdater funktionen skriver vi følgende:

*#Vi tilføjer kaktussens opdater funktion til kaktus dictionarien*

kaktus['opdater'] = kaktus\_opdater

Og efter fugl\_opdater funktionen skriver vi:

*#Vi tilføjer fuglens opdater funktion til fugl dictionarien*

fugl['opdater'] = fugl\_opdater

Nu skal vi bare kalde den i vores inderste game loop, i stedet for de andre opdater funktioner skriver i derfor:

…

**while** dino['levende']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hopper'] = True

dino['hop lyd'].play()

*#Opdater farten*

spil['fart'] += 0.001

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Opdater fjenden*

spil['fjende']['opdater']()

*#Tegn spillet*

tegn()

…

Vi skal også lige erstatte de ”reset” dictionaries til vores fugl og kaktus vi har i det yderste gameloop. Så de kommer til at have ”opdater” værdien med sig, vi ændrer dem fra det her:

kaktus = {'x':bredde, 'y':højde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5), 'hoppet over':False}

fugl = {'x':bredde, 'y':højde-81-10, 'bredde':94, 'højde':81, 'billed\_nr':0, 'hoppet over':False}

til det her:

kaktus = {'x':bredde, 'y':højde-70-10, 'bredde':34, 'højde':70, 'billed\_nr':random.randint(0, 5), 'hoppet over':False, 'opdater':kaktus\_opdater}

fugl = {'x':bredde, 'y':højde-81-10, 'bredde':94, 'højde':81, 'billed\_nr':0, 'hoppet over':False, 'opdater':fugl\_opdater}

Hvorfor ændrer fjenden sig ikke? Prøv at lukke spillet ned og åben det op igen. Spøjst ikke.

## Step 16: Ja, hvorfor ændrer fjenden sig ikke

Fjenden ændrer sig ikke fordi vi ikke beder den om at ændrer sig! Så det bliver vi nødt til at bede den om. Hvornår skal vi bede den om at ændrer sig? Det skal vi efter fjenden har ramt venstre side af skærmen, derfor til føjer vi den samme linje som vi definerede ’fjende’ værdien med nemlig:

spil['fjende'] = [kaktus, fugl][random.randint(0,1)]

Den vil vi tilføje både i kaktus\_opdater og i fugl\_opdater så de kommer til at se således ud:

**def** kaktus\_opdater():

*#Flyt kaktussen*

kaktus['x'] -= 20\*spil['fart']

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** kaktus['x'] <= 0:

kaktus['x'] = bredde + random.randint(0, 200)

kaktus['billed\_nr'] = random.randint(0, 5)

kaktus['hoppet over'] = False

spil['fjende'] = [kaktus, fugl][random.randint(0,1)]

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > kaktus['x']+kaktus['bredde'] **and** **not** kaktus['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

kaktus['hoppet over'] = True

Og sådan her:

**def** fugl\_opdater():

*#Flyt kaktussen*

fugl['x'] -= 20\*spil['fart']

*#Skift fuglens billede*

**if** fugl['billed\_nr'] >= 1.5:

fugl['billed\_nr'] = 0

**else**:

fugl['billed\_nr'] += 0.4

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** fugl['x'] <= 0:

fugl['x'] = bredde + random.randint(0, 200)

fugl['billed\_nr'] = 0

fugl['hoppet over'] = False

spil['fjende'] = [kaktus, fugl][random.randint(0,1)]

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > fugl['x']+fugl['bredde'] **and** **not** fugl['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

fugl['hoppet over'] = True

Prøv det, tænk sådan en lille ændring kan gøre så meget. Husk at computeren altid kun gør hvad du beder den om.

## Step 17: Kunne den ikke også dukke sig?

I det rigtige spil kunne vores dino da også dukke sig, skal vi ikke også have det med? RETORISK spørgsmål som jeg nu vil svarer på: For jo selvfølgelig skal vi det!!

Men for at få den til at dukke sig skal vi bruge nogle andre billeder og dermed også nogle andre dimensioner, vi skal derfor tilføje en dictionary til at indeholde disse værdier. Derudover skal vi også vide om dinoen skal dukke sig lige nu, de kommer dermed til at sådan her ud:

*#Dino billedet og dino variablen der indeholder værdierne knyttet til dinoen og det samme men for når den dukker sig*

dino\_billede = pygame.image.load("dinoer.png")

dino\_duk\_billede = pygame.image.load("dinoduk.png")

dino = {'x':120, 'y':højde - 95 - 10, 'bredde':440//5, 'højde':95, 'billed\_nr':2, 'hopper':False, 'v':8, 'm':2, 'y\_min': højde - 95 - 10, 'hop lyd':pygame.mixer.Sound("hop.wav"), 'levende':True, 'dukker':False}

dino\_duk = {'x':120, 'y':højde-60-10, 'bredde':236/2, 'højde':61, 'billed\_nr':0}

Nu skal vi sørge for at dino\_duk bliver opdateret hvis dinoen dukker sig, det betyder at vi skal spørge om den dukker sig eller ej, hvis ja så skal vi opdatere dino\_duks billed\_nr ellers skal vi bare opdatere den anden. Dino\_opdater funktionen kommer til at se sådan her ud:

**def** dino\_opdater():

*#Har dinoen ramt kaktussen*

dino\_ramt(kaktus['x'], kaktus['y'], kaktus['bredde'], kaktus['højde'])

*#Har dinoen ramt fuglen*

dino\_ramt(fugl['x'], fugl['y'], fugl['bredde'], fugl['højde'])

*#Hopper dinoen*

**if** dino['hopper']:

dino\_hopper()

*#Skift dinoens animation*

**if** dino['dukker']:

**if** dino\_duk['billed\_nr'] >= 1.5:

dino\_duk['billed\_nr'] = 0

**else**:

dino\_duk['billed\_nr'] += 0.4

**else**:

**if** dino['billed\_nr'] >= 3.5:

dino['billed\_nr'] = 2

**else**:

dino['billed\_nr'] += 0.4

Når vi dukker os er det fordi vi vil under noget, men så går det ikke vi dør fordi vores program tror vi forbliver samme højde. Vi ændrer derfor dino\_ramt funktionen så den ved om vi dukker os eller ej, den ender med at se sådan her ud:

**def** dino\_ramt(x, y, b, h):

**if** **not** dino['dukker'] **and** dino['x']+dino['bredde'] > x **and** dino['x'] < x+b **and** dino['y']+dino['højde'] > y **and** dino['y'] < y+h:

dino['levende'] = False

**elif** dino['dukker'] **and** dino\_duk['x']+dino\_duk['bredde'] > x **and** dino\_duk['x'] < x+b **and** dino\_duk['y']+dino\_duk['højde'] > y **and** dino\_duk['y'] < y+h:

dino['levende'] = False

Vi skal også tegne vores dino når den dukker sig, vi tilføjer ligesom de andre steder et if-statement, der spørger om dinoen dukker sig eller ej og tegner på baggrund det rigtige. Tegn funktionen kommer til at sådan her ud:

**def** tegn():

*#Tegn hvid baggrund*

skrm.fill((255,255,255))

*#Tegn dinoen*

**if** dino['dukker']:

skærm.blit(dino\_duk\_billede, (dino\_duk['x'], dino\_duk['y']), (dino\_duk['bredde']\*int(dino\_duk['billed\_nr']), 0, dino\_duk['bredde'], dino\_duk['højde']))

**else**:

skærm.blit(dino\_billede, (dino['x'], dino['y']), (dino['bredde']\*int(dino['billed\_nr']), 0, dino['bredde'], dino['højde']))

*#Tegn jorden, først den ene så den anden*

skærm.blit(jord\_billede, (jord['x1'], jord['y']))

skærm.blit(jord\_billede, (jord['x2'], jord['y']))

*#Tegn kaktussen*

skærm.blit(kaktus\_billede, (kaktus['x'], kaktus['y']), (kaktus['bredde']\*kaktus['billed\_nr'], 0, kaktus['bredde'], kaktus['højde']))

*#Tegn fuglen*

skærm.blit(fugl\_billede, (fugl['x'], fugl['y']), (fugl['bredde']\*int(fugl['billed\_nr']), 0, fugl['bredde'], fugl['højde']))

*#Tegn scoren*

visScore()

*#Vis det der er tegnet på skærmen*

pygame.display.flip()

*#Vent 30 milisekunder*

pygame.time.delay(30)

Nu mangler vi bare at lade spilleren styrer hvornår, dinoen skal dukke sig og skal komme op igen, vi skal derfor ned til det inderste gameloop for at tilføje yderligere mulige tastetryk. Derudover skal vi også huske at få dinoen til at stoppe med at dukke sig, hvis vi slipper den tast. Det inderste game loop ender med at se sådan her ud:

**…**

**while** dino['levende']:

*#For hver begivenhed(tastatur tryk eller lignende)*

**for** event **in** pygame.event.get():

*#Er der blevet trykket på 'X' i højre hjørne?*

**if** event.type == pygame.QUIT:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Er der trykket en tast ned?*

**if** event.type == pygame.KEYDOWN:

*#Hvis ja, er denne tast escape-tasten?*

**if** event.key == pygame.K\_ESCAPE:

*#Hvis ja, stop spillet*

spil['kør'] = False

*#Hvis pil op bliver trykket ned skal dinoen hoppe*

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

dino['hopper'] = True

dino['hop lyd'].play()

*#Hvis pil ned bliver trykket ned skal dinoen dukke*

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

dino['dukker'] = True

**if** event.type == pygame.KEYUP:

**if** event.key == pygame.K\_DOWN:

dino['dukker'] = False

*#Opdater farten*

spil['fart'] += 0.001

*#Opdater jordens position*

jord\_opdater()

*#Opdater dinoen position*

dino\_opdater()

*#Opdater fjenden*

spil['fjende']['opdater']()

*#Tegn spillet*

tegn()

…

Prøv det kan den dugge sig : )

## Step 18: Hvorfor var det den skulle dukke sig?

Hvorfor var det egentligt at vores dino skulle kunne dukke sig? Det var jo selvfølgelig fordi den skulle kunne løbe under fuglene, men før den kan det skal vi lige sørge for at nogle af fuglen kommer op i luften. Det kan vi gøre ved tilfældigt at hæve den præcis med den højde vores dino har når den dukker sig (61 pixels). Vi kan dermed ændre vores fugl dictionary til følgende:

fugl = {'x':bredde, 'y':højde-81-10-61\*random.randint(0,1), 'bredde':94, 'højde':81, 'billed\_nr':0, 'hoppet over':False}

Nu er det kun den første fugl, der har 50% chance for at flyve højt, lad os lige give de andre fugle den samme chance. Vi ændrer derfor fugl\_opdater funktionen, så den også resetter y værdien hver gang fuglen har nået den anden side af skærmen. Den kommer til at se sådan her ud:

**def** fugl\_opdater():

*#Flyt kaktussen*

fugl['x'] -= 20\*spil['fart']

*#Skift fuglens billede*

**if** fugl['billed\_nr'] >= 1.5:

fugl['billed\_nr'] = 0

**else**:

fugl['billed\_nr'] += 0.4

*#Hvis kaktussen ryger ud for skærmen, smid den tilbage og vælg et tilfældigt billede*

**if** fugl['x'] <= 0:

fugl['x'] = bredde + random.randint(0, 200)

fugl['y'] = højde-81-10-61\*random.randint(0,1)

fugl['billed\_nr'] = 0

fugl['hoppet over'] = False

spil['fjende'] = [kaktus, fugl][random.randint(0,1)]

*#Hvis spilleren er på høre side af kaktussen og han ikke allerede har hoppet over den så*

**if** dino['x'] > fugl['x']+fugl['bredde'] **and** **not** fugl['hoppet over']:

*#Læg 1 til spillerens score*

spil['score'] += 1

*#Gør sådan at kaktussen er blevet hoppet over*

fugl['hoppet over'] = True

Prøv det, ligner det ikke meget godt det endelige spil?

## Step 19: High scoren revideret

Det går ikke at spillet ikke kan gemme min high score, jeg fik altså 15 point! Nu skal vi sørge for at den gemmer det på tværs af tid, det kan vi gøre ved at gemme det i en fil. I python kan vi arbejde med en lang række forskellige fil typer (.csv …), vi vælger at kigge på .txt fil typen som er ren tekst, så den er nem at gå til. I python skal man åbne en fil, før man kan skrive til eller læse fra den, man åbner en fil sammen med den måde man gerne vil kommunikere med filen på, vi vil bruge følgende ”modes” ”w”:(write) hvis filen ikke eksisterer laver den filen og skriver, ellers skriver den over, ”r”:(read) kan læse fra filen.

Så vi vil gerne starte med at læse værdien fra den .txt fil hvor vi gemmer vores highscore. Vi skal derfor lige lave en .txt fil, skriv 0 i filen og gem den som highscore.txt. Så skal du så skrive følgende:

spil = {'kør':True, 'score':0, 'high score':0, 'fart':1, 'high score fil': 'highscore.txt'}

*#Åben high score filen og læs tallet derfra, luk filen*

**with** open(spil['high score fil'], 'r') **as** fil:

spil['high score'] = int(fil.read())

fil.close()

Nu skal vi også sørge for at skrive scoren til filen, hvis den er højere en high scoren. Det gør vi der hvor vi ellers ville opdatere high scoren i det yderste game loop.

…

*#Hvis scoren er større en high scoren opdater high scoren*

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

*#Skriv high scoren til high score filen*

**with** open(spil['high score fil'], 'w') **as** fil:

fil.write(str(spil['score']))

fil.close()

spil['score'] = 0

Sådan prøv at spil og så luk spillet ned, gemmer den det? Kan du gå ind og ændrer scoren? Hvor nemt er det egentlig at ændrer high scoren til noget helt urealistisk?

## Step 20: Ej så nemt må det heller ikke være.

Når det bare var et tal, der blev gemt skulle der ikke så meget til før det kunne hackes. Det vil vi gerne undgå. Lad os prøve at gøre det lidt sværere ved bruge noget, der hedder binære tal. Binære tal er det man kalder base 2, fordi de kun har to forskellige symboler (0 og 1) til at udtrykke værdien med, hvor vores normale tal er base 10 fordi der er ti forskellige symboler til at udtrykke værdien med (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Det vi gør er at vi ændrer scoren til det ækvivalente binære tal og skriver det til filen. Så der hvor den sidste redigering førte sted i det yderste game loop, skal vi nu ændrer det til det følgende:

…

*#Hvis scoren er større en high scoren opdater high scoren*

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

*#Lav highscoren til et binært tal, skriv high scoren til high score filen*

**with** open(spil['high score fil'], 'w') **as** fil:

fil.write(str(bin(spil['score'])))

fil.close()

spil['score'] = 0

Ps. Ændrer det der står i highscore.txt filen til 0b0. Det betyder også at vi skal læse tallet på en anden måde nu hvor det er et binært tal. Vi ændrer koden oppe ved dictionariesne til følgende:

*#Åben high score filen og læs tallet derfra, luk filen*

**with** open(spil['high score fil'], 'r') **as** fil:

*#det andet argument i int funktionen er hvilken base taller er i*

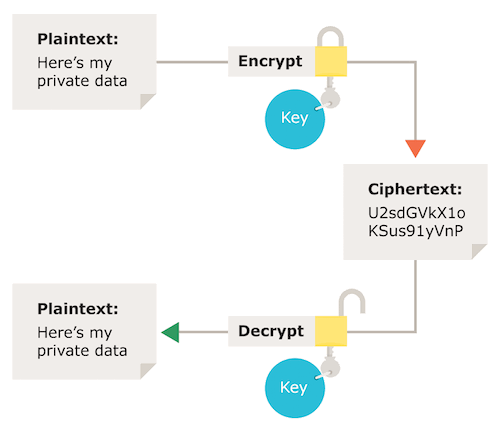
spil['high score'] = int(fil.read(), 2)

fil.close()

Spil en omgang og luk så spillet ned. Prøv nu at åbne filen kan du forstå hvad der stå? Kan du ændrer teksten så scoren bliver højere?

## Step 21: Ingen hackers her tak!

Det gjorde det ikke meget sværere med de binære tal. Nej nu stopper det simpelthen, vores score på ikke kunne blive hacket! Lad os enkryptere den! Kryptering bruger man blandt andet når to personer skriver til hinanden, i stedet for at beskeden bare flyver gennem luften bogstav for bogstav så alle kan lytte til den, bliver den krypteret. Det betyder at en eller anden algoritme laver en helt uforståelig besked ved hjælp af en såkaldt nøgle, der deles med den anden person. Når den anden person har beskeden og nøglen kan den person bruge nøglen til at ”åbne” for den krypterede besked fordi man skal bruge nøglen til at få gendannet beskeden.



Billedet ovenover illustrerer det ret fint.

Det vil vi også gerne gøre med vores besked, men i stedet for at skrive det hele fra bunden får vi lidt hjælp. Ligesom vi bruger biblioteket pygame vil vi nu til at bruge et krypterings bibliotek, der hedder simple-crypt. Men før vi kan bruge det skal vi lige downloade det, det gør vi ved at skrive følgende kommando i vores kommandoprompt eller terminal:

python3 -m pip install simple-crypt

Når den er downloadet skal vi have det ind i vores program, vi skal kun bruge to af bibliotekets funktioner ind i vores program, vi importerer det derfor således:

**from** **simplecrypt** **import** encrypt, decrypt

Vi skal nu ændrer måden vi indlæser vores high score første gang til følgende. Men først skal vi live lave en anden tekst fil ved navn ” highscorekey.txt” og sørg for at der ikke står noget i den, der må heller ikke stå noget i highscore.txt filen. Ændrer derefter koden til følgende:

*#Spil variablen indeholder alle de værdier der er knyttet til spillets kørsel*

spil = {'kør':**True**, 'score':0, 'high score':0, 'fart':1, 'high score fil': 'highscore.txt', 'fil nøgle' : 'highscorekey.txt'}

*#Åben high score nøgle filen og læs tallet derfra*

**with** open(spil['fil nøgle'], 'r+') **as** filnøgle:

nøgle = filnøgle.read()

*#Hvis der ikke er nogen nøgle så lav en nøgle og sæt high scoren til 0*

**if** nøgle == "":

filnøgle.write(str(random.randint(0,2000)))

spil['highscore'] = 0

*#Hvis der er en nøgle så åben high score filen, læs fra den fil og decryptér det der står i den.*

**else**:

**with** open(spil['high score fil'], 'rb') **as** fil:

spil['high score'] = int(decrypt(str(nøgle), fil.read()).decode(), 2)

fil.close()

filnøgle.close()

Nu skal vi også huske at ændrer der hvor vi skriver til highscore.txt filen, så det også bliver enkrypteret. Vi ændrer koden i det yderste game loop til det følgende:

*…*

*#Hvis scoren er større en high scoren opdater high scoren*

**if** spil['score'] > spil['high score']:

spil['high score'] = spil['score']

*#Lav en ny nøgle og skriv den til nøgle filen*

**with** open(spil['fil nøgle'], 'w') **as** filnøgle:

nøgle = random.randint(0, 2000)

filnøgle.write(str(nøgle))

*#Åben high score filen, enkrypter den binære værdi, skriv den derefter til high score filen.*

**with** open(spil['high score fil'], 'wb') **as** fil:

fil.write(encrypt(str(nøgle), str(bin(spil['score']))))

fil.close()

filnøgle.close()

spil['score'] = 0

Spil spillet lidt. Prøv at se på hvad filerne har i sig, er det ikke lidt sværere at forstå. Prøv at ændrer det første tal, hvad sker der?