ABB Industrigymnasium

Joakim Flink 2019-02-04

AI-mus

Vad vi ska lära oss:

1. Skapa 2d array
2. Skapa grafiska objekt i python
3. Göra en lättare AI

Innehåll

Definition

I denna uppgift ska vi göra en 2d värld där en mus ska leta efter ost. Detta för att träna oss på Python och bli bättre på metoder, loopar och ifsatser. Vi kommer att lära oss att använda grafiska

biblioteket tkinter och grunderna till AI. Dessa grunder kan ni snegla på senare i ert robotprojekt för att ha en bra utgångpunkt i ert arbetssätt.

Uppgifter

1) Skapa värld

För att skapa världen vill vi göra det i två steg, Först datamässigt, vart ska osten ligga, vart är det väggar och vart är det öppen mark. Detta görs i form av en 2D array där vi även kommer kunna hålla koll på vart musen är och på detta sätt flytta runt honom i världen. Utifrån denna 2d "karta" så ritar vi upp världen och uppdaterar den.

1) 2D array

För att skapa världen data mässigt finns det två sätt, hårdkodat och dynamsikt. Vi kommer gå igenom bägge

(1) Hårdkodat

```
map1=([[ 0,0,0,0],
       [ 0,1,0,0],
       [ 0,0,0,0],
       [ 0,0,0,0],
       [ 0,0,0,0]])
```

På detta sätt skapar du en 2d array som innehåller 4 * 4 värden. Vill jag att osten ska ligga på rad 2, plats 2 så har lagt en 1a där. I detta fall betyder inte ettorna och nollorna någonting, det är sen när vi ritar ut världen vi bestämmer vad de olika tecknen ska ha för betydelse.

(2) Dynamiskt

Det andra sättet vi kan göra detta på är dynamiskt. Vad dynamiskt betyder i detta fall är att det är flexibelt och anpassningsbart men framförallt är att det är automatiserat till skillnad från hårdkodat.

```
w = 10
h = 10
Map = [[0 for x in range(w)] for y in range(h)]
food= 1
wall=2
Map[2][2]= food
Map[0][1]= wall
```

På detta sättet skapar vi 2D arrayen genom att först definera hur många rutor det ska vara, först i bredd (w) sedan i höjd(h). Man ser ganska snabbt att det här är smidigare för att skapa större rutnät, som det vi gör nu är 10*10 istället för 4*4 som vi orkade skriva manuellt. Vi skapar variabeln Map och säger att den ska lägga 0 på tio platser inom de första hårda parantesarna och samma för de andra.

Det vi senare gör är att vi skapar variablerna food och wall med deras respektive värde och lägger dem på den platsen i arrayen där vi vill ha dem, man skriver alltså över det tidigare 0 värdet som låg där.

För att se vad som ligger på en viss plats så skriver du enkelt:

```
print(map1[0][0])
print(Map[0][0])
```

2) Generera värld

Förarbete:

Lägg dessa två rader högst upp i koden, den första laddar in tkinter, som är vårt biblitek för att skapa grafiska objekt. Den andra raden är att vi säger att TK(Tkinter) ska fungera i hela programmet utan att vi behöver skapa separata tkinter objekt.

```
from tkinter import *
master = Tk()
```

Det första vi ska göra är att skapa vår Canvas, det vill säga vår värld.

```
C = Canvas(master, bg="green", height=600, width=600)
C.pack()
```

Här säger vi att vi vill ha grön bakgrundsfärg och storleken 600 *600 pixlar. Vi packar sedan in canvasen i vårt masterobjekt så den visas på skärmen. Testa gärna att köra koden, både med och utan C.pack(), för att göra detta så måste du lägga till mainloop() längst ner i koden.

Nu kommer vi till delen då vi ska skapa rutnätet. Nu kommer man vara tvungen att tänka logiskt, klura lite själv hur du hade velat gjort det innan du kollar på nästa steg.

(1) Skapa världen!

Först måste vi planera vi vet att 10*10 objekt ska ut på skärmen och vi vill varken ha de på samma plats allihopa eller att de är för små eller för stora.

```
distanceY=0
distanceX=0
sizevalue=50
```

Därför skapar vi variabler, först skapar vi distY och X för att ha koll på distans mellan de olika objekten som skapas så varje får en unik plats. Den tredje variabeln heter sizevalue och där kommer man kunna labba för att se till att de blir den storlek man önskar.

Nu kommer skaparen av världar, och som några av er säkert gissade så gör vi det dynamsikt med forloopar. Här är det två foorloopar vid varandra, dessa kallas för nässlade forloopar. Som de funkar i detta fall så går första, som vi kallar row igenom alla världen från 0 till w(10) och inuti den så gör col likadant för 0 till h(10).

Här inne skapar vi ett objekt som vi kallar coord(kordinat) där vill vi ha fyra värden, Dessa värden är startvärden för x, y samt startvärdena plus storleken. Detta resulterar i att vi får de fyra punkterna som är vår rektangels hörn.

```
for row in range(0,w):
    for col in range(0,h):
       coord = distanceX, distanceY, distanceX+sizevalue, distanceY+sizevalue
#y,x, w, h
       distanceX+=sizevalue
       if Map[col][row]==1:
              C.create_rectangle(coord, fill="yellow")
       elif col%2==0:
              C.create_rectangle(coord, fill="white")
       elif col%2==1:
              C.create_rectangle(coord, fill="gray")
    distanceY+=sizevalue
    distanceX=0
```

När vi har satt coord så uppdaterar vi distanceX så nästa rektangel inte hamnar på samma plats som den förra. Detta gör vi genom att addera storleken till distanceX.

Nu tittar vi om värdet på plats Map[col][row](alltså 0,0 första varvet) är 1, Är detta fallet så ritar den ut en gul rektangel på kordinaterna. På liknande sätt kan ni göra om ni vill ha med fler objekt som väggar och annat.

Är det inte så kollar vi om col är ett jämnt eller ojämnt nummer med hjälp av modelus och ritar ut planen som ett fint rutnät. För att få distancen att ändras i Y ledt så uppdaterar vi distanceY med sizevalue när col har gått hela sin loop en gång, alltså i vårt fall efter 10 värden har ritats ut.

För att få till rutnätet så får vi såklart inte glömma att ändra tillbaka startvärdet för X till 0!

Pangprego så var det klart, det ända vi behöver göra nu är att lägga till mainloop längst ner så att världen ritas ut.

```
mainloop()
```

2) Lägg till mus

Vi ska i denna del skapa en mus, skapa en finare mus(kodmässigt) och få den att röra sig.

1) Skapa mus

För att skapa musen gör vi ett arrayobjekt som tar in två värden, musens x och y plats i 2d arrayen.

Vi skapar musens grafiska x position genom att ta mousevariabelns första värde(x värdet) och multiplicerar den med sizevalue. Likadant görs för y och efter det lägger vi in som ni tidigare gjort dessa värden i en coord variabler.

Efter detta skapar vi en till rektangel, denna döper vi till Mouse för att kunna ändra den sen.

```
# Skapa musen
mouse= [5,5]
x= mouse[0]*sizevalue
y=mouse[1]*sizevalue
coord = x, y,x +sizevalue, y+sizevalue
Mouse=C.create_rectangle(coord, fill="brown")
```

2) Fina till musen

För att fina till musen ska vi bli lite objektorienterade och minska ner raderna genom att anvädna metoder. Vi börjar med att gå till toppen av koden, precis efter vi har skapat masterobjektet.

Här skapar vi två metoder genom att skriva def. Den första kallar vi rs för rightsize, den tar in två inputs, inputval och sizevalue. Det den kort och gott gör är att den får in ett värde och räknar ut dess start kordinater.

```
#rightsize
def rs(inputval,sizevalue):
	return inputval*sizevalue
```

Den andra metoden kallar vi för getcoord, här skickar vi in x och y värde (men man kan om man vill även skicka in size om man modifierar den lite). X och ys kordinater sätts genom att ta x och y platsen i rutnätet och multiplicera den med sizevalue md hjälp av rs metoden. Detta returnerar den sen som en coord variabel,

```
# Get coordinates, send in x and y value and return them + the size  of the
object
def getcoord(x,y):
	sizevalue=50
	x= rs(x,sizevalue)
	y= rs(y,sizevalue)
	return x, y,x +sizevalue, y+sizevalue
```

Simsalabim så är vår skapa mus nu bara två rader.

```
# Skapa musen v2
mouse= [5,5]
Mouse=C.create_rectangle(getcoord(mouse[0],mouse[1]), fill="brown")
```

Om man egentligen vill kan den till och med bara vara en rad

```
Mouse=C.create_rectangle(getcoord(5,5), fill="brown")
```

3) Flytta musen

För att flytta ett objekt så finns det flera sätt, du kan rita ut objektet på en ny plats och sen använda update till exempel. Men i detta fall ska vi använda metoder move().

```
C.move(Mouse,mouse[0]*sizevalue,mouse[1]*sizevalue)
```

Om vi lägger in den här koden och kör programmet så ser vi att vår mus har flyttat sig. Men stämmer flyttningen verkligen? Det vi har sagt är att på Canvasen ska vi flytta rektangelobjektet Mouse från musens nuvarande plats till egentligen samma plats men ändå har musen flyttat sig. Detta beror på att vi har sagt åt musen att flytta sig ytterliggare 5 platser åt höger och 5 platser ner. Alltså kan vi inte bara skriva in 2Darrayens kordinater utan vi måste få in skillnaden från nuvarande plats och nästa. Detta ska vi göra nu.

0,0	0.1	0.2	0.3
1.0	1.1	1.2	1.3
2.0	2.1	2.2	2.3
3.0	3.1	3.2	3.3

Rutnät 1 Om jag är på plats 1.0 och ska till 1.1 så kan jag inte skriva in 1.1 för då rör jag mig 1.1 avstånd och hamnar på 2.1

Ta bort C.move raden och lägg istället till den nedan, där vi skapar en array med olika kommandon som förklarar för musen hur hen ska röra sig.

```
mousemove=["upp","upp","vänster","upp","upp","vänster"]
```

Det vi nu ska göra är att göra rörelsen smart direkt och skapar därför en def högst upp i koden. Vi importerar även time innan dess.

```
import time
def Movemouse(mouse1,Movingobject):
	time.sleep(1)
	dx= mouse1[0]*sizevalue
	dy= mouse1[1]*sizevalue
	C.move(Movingobject, dx, dy)

	C.update()
```

Vi skickar in ett objekt som vi kallar mouse1, det är en array med två värden x på plats 0 o y på plats 1. Vi skickar även in Movingobject som är Mouse objektet, alltså det vi vill flytta på. Här lägger vi en time.sleep() på en sekund för att hinna se förflyttningen. Det som kommer senare är bekant redan, vi skapar kordinater utifrån 2darray positionerna multiplicerat med sizevalue. Nu frågar ni er själva, men blir inte det här samma som innan? Jo, beroende på vad man skickar in, därför är vi nu på nästa steg.

Efter mousemove raden skriver du följande kod:

```
while len(mousemove)>0:
	print(str(mouse[0])+"  "+str(mouse[1])) #Så man ser vad som händer
	steg=[0,0]
	C.create_rectangle(getcoord(mouse[0],mouse[1]), fill="blue")
	if mousemove[0]=="upp":
		mouse[1]= mouse[1]-1
		steg[1]-=1
	elif mousemove[0]=="ner":
		mouse[1]= mouse[1]+1
		steg[1]+=1
	elif mousemove[0]=="vänster":
		steg[0]-=1
		mouse[0]= mouse[0]-1
	elif mousemove[0]=="höger":
```

```
            steg[0]+=1
            mouse[0]= mouse[0]+1
        Movemouse(steg,Mouse)
        mousemove.pop(0)
```

Vad den här koden gör är att vi tittar sålänge mousemove har fler än 0 värden så ska detta köras. Vi börjar med att skriva ut nuvarande positionen med en print, detta för att se att han rör sig och rör sig rätt. Efter detta så skapar vi än en gång en array som innehåller 2 värden, dessa kallar vi steg och den börjar som tom alltså [0,0]. Där musen står just nu skapar vi blå rektangel, detta för att visa hur han rör sig. **Denna del kan ni ta bort om ni inte vill ha eller göra mer avancerad så det senaste steget är stark blått men det innan är svagare och svagare...**

Vi tittar vad som ligger i mousemove på plats 0, är det "upp så lägger vi in -1 i på y värdet, vi gör även detta på mouse för att kunna ge detta värde till vårt blåa spår nästa varv i loopen. Är det något av de andra så händer liknande sak i deras fall, På nästsista raden körs Movemouse med stegen(ex [0,-1]) istället för musens position(ex [3,2]) och därav får vi musens förflyttning istället för hela positonen. Det sista vi gör är använda pop för att ta bort värde 0 i mousemove och det tidigare andra mousemove värdet blir nu det första.

TESTA

3) Hitta ost

I detta steg ska vi göra det möjligt för musen att hitta ost, Detta ska vi göra genom att det skrivs ut i ett textfönster när han hittar ost och även när han har slut på steg/enegi.

1) Textruta

För att skapa en textruta så skapar vi än en gång ett barn object till det grafiska objektet, denna heter Text och tar in höjd och längd. Vi lägger även in en testtext som ska skrivas ut. Man kan antingen välja att texten ska läggas i början eller slutet av den redan befintliga texten, alltså så kan du bygga vidare på texter över tid.

```
text = Text(master, height=2, width=30)
text.pack()
text.insert(END, "Testtext \nÖver två rader\n")
```

Vi kommer komma tillbaka till Text lite längre ner i denna del men vill man lära sig mer kan man titta på https://www.python-course.eu/tkinter_text_widget.php .

2) Finna osten

Vi ska börja med att vi gör en ny variabel högst upp i koden, denna ska ha koll på hur mycket ost som samlas in under musens gång och denna kallas för global för att den ligger fritt att användas över hela koden.

```
cheeseeaten=0
```

Vi ska nu skapa en ny funktion, denna kallar vi FoundCheese och den vill ha in en XY position i rutnätet. Detta för att kolla om det är en 1a alltså ostobejkt på den platsen. Är det så att det är en ost på platsen så skriver den ut att den hittat osten och på vilken plats den fanns. Str() gör om nummervariabler till strings. Vi hämtar den globala variabeln cheeseeaten och adderar en till den.

```
def FoundCheese(mousepos):
    if Map[mousepos[0]][mousepos[1]]==1:
        outputtext= "Found Cheese at"+str(mouse[0])+" X  "+str(mouse[1])+ " Y"
        global cheeseeaten
        cheeseeaten+=1
        text.insert(INSERT, outputtext)
```

Denna funktion vill vi sedan anroppa efter Movemouse och innan mousemove.pop för att kunna titta där musen är just nu.

```
        Movemouse(steg,Mouse)
        FoundCheese(mouse)
        mousemove.pop(0)
```

För att ha ett litet mysigare/mer informativt slut på musens resa så lägger vi in text.delete för att ta bort all text och sedan lägger vi in en ny som berättar hur mycket ost som har ätits.

```
text.delete("insert linestart", "insert lineend")
text.insert(END, "Out of steps\n Cheese eaten "+str(cheeseeaten))
```

4) Musen lever!

I detta steg ska vi se till att musen lever av egen kapacitet, närmare bestämt kan röra sig själv runt fritt på spelplanen.

1) Dum mus

Första nivån är en dum mus, en väldigt dum mus. Hans livsval baseras på slumpmässiga val och endast det, För att göra detta möjligt så måste vi kunna få fram slumpmässiga värden, detta gör vi genom att importera randint högst upp.

```
from random import randint
```

Sen som ni säkert kunde ana så ska vi göra ännu en funktion denna kommer vi kalla för MovePlanner även om vi inte planerar så mycket. Det den ska göra är att skapa en array med instruktioner om hur den ska röra sig, se exempel nedan.

["upp","upp","vänster","vänster","vänster","upp","upp","höger"]

Fundera en stund först hur du skulle kunna skapa detta, titta sedan igenom koden nedan och se hur mycket du förstår/det liknar din lösning.

```
def MovePlanner(steps):
    steglista=[]
    while steps>0:
        typavsteg=randint(0, 3)
        if typavsteg==0:
            steglista.append("upp")
        elif typavsteg==1:
            steglista.append("ner")
        elif typavsteg==2:
            steglista.append("vänster")
        elif typavsteg==3:
            steglista.append("höger")
        steps-=1
    return steglista
```

Vad gör funktionen då? Jo den ger ut en array med instruktioner utifrån att endast veta hur många steg den ska gå. Detta kan vi ersätta mousemove variabelns värden så det istället ser ut som någon av raderna nedan. Vad är det för skillnad på dem, testa!

```
mousemove=MovePlanner(3)
mousemove=MovePlanner(randint(3,10))
```

2) Mindre dum mus

För **mindre dum mus**, eller egentligen hens efterföljare **helt okej mus** så ska vi göra lite förarbete genom att lägga upp musens övergrippande variabler högt upp i vår kod. Förutom att lägga till två arrayer, korttidsminne och långstidsminne och värdet energi så flyttar vi upp mouse, som innehåller musens position upp hit.

```
#musdetaljer
korttidsminne=[]
långtidsminne=[]
energi=50
mouse= [1,2] #position
```

Därefter ska vi ändra lite i vår Movemouse för om ni inte redan märkt det så är det en liten miss i vår kod, man kan gå ut från planen. Detta ska vi nu förhindra genom att vi anropar den globala mouse som har koll på positonen musen är på just nu. Det vi gör är att vi kollar om den nuvarande positionen är **större** än 0 i x och y led samt mindre än h(eight) och w(idth). Är det utanför banan så ändras mouse1 (rörelseförändringen ex 1 eller -1) så den blir 1 om den var 0 och viceversa. Även mouse (positionen) ändras och då med +2 eller -2 på grund av att vi redan ändrat denna i vår while loop. *Det här är inte ett optimalt sätt att göra detta på men det är ett bra exempel på hur projekt blir om man bygger på dem utan att ha en tydlig plan och struktur från grunden.*

```
def Movemouse(mouse1,Movingobject):
       time.sleep(1)
       global mouse
       if mouse[0]<=-1:
           mouse1[0]=1
           mouse[0]= mouse[0]+2
       elif mouse[1]<=-1:
           mouse1[1]=1
           mouse[1]= mouse[1]+2
       elif mouse[0]>=w:
           mouse1[0]=-1
           mouse[0]= mouse[0]-2
       elif mouse[1]>=h:
           mouse1[1]=-1
           mouse[1]= mouse[1]-2
       dx= mouse1[0]*sizevalue
       dy= mouse1[1]*sizevalue
       C.move(Movingobject, dx, dy)
       C.update()
```

Som sista del så gör vi musen lite mindre dum genom att han kommer ihåg och kollar åt vilket håll han gick senast. Detta gör vi genom att skapa en variabel som heter formerstep, den har koll på vilket det förra steget var, Vi adderar ett and-argument till våran if-elif sats vilket betyder att man kollar om två värden stämmer. Alltså om **typavsteg==1 och förrasteget != (inte likamed) 0** .

Det sista vi behöver göra är att titta så det inte är samma steg, är det samma steg som förra gången så ska vi låta loopen köras ett varv till innan vi tar bort ett steg. Det är viktigt att formersteps sätts inuti den här ifsatsen annars kan formerstep bli en typ som den inte borde vara.

```
def MovePlanner(steps):
    steglista=[]
    formerstep=5
    while steps>0:
        samma=False
        typavsteg=randint(0, 3)
        if typavsteg==0 and formerstep!=1:
            steglista.append("upp")
        elif typavsteg==1 and formerstep!=0:
            steglista.append("ner")
        elif typavsteg==2 and formerstep!=3:
            steglista.append("vänster")
        elif typavsteg==3 and formerstep!=2:
            steglista.append("höger")
        else:
            samma=True
```

```
        if samma==False:
            formerstep=typavsteg
            steps-=1
    return steglista
```

5) Helt ok mus

I helt ok mus ska vi använda lång & kortidsminnet vi la in samt energin. Kortidsminnet ska ha koll på stegen han tar medan långtidsminnet har koll på vart han har hittat ost. Energin är energin som han har till sitt förfogande att förbruka 1 energi går åt varje gång han rör sig och om han hittar ost får han tillbaka 10 energi. Detta kommer bli den sista och mest omfattande delen i vårt program och kommer därför delas upp i bitar. När man är klar med denna del får man antingen gå vidare med livet eller förbättra musen genom att exempelvis:

- göra den smartare
- lägga till fler möss
    - göra så att möss kan föröka sig
- ge den synfält
- göra det möjligt för musen att bli bättre med tiden
    - röra sig längre
    - konsumera mindre energi
    - med mera
- Fixa buggar (det finns några gömda i koden som ni säkert märkt)
- Mycket annat

1) Nya globala variabler

Det första vi gör är att skapar de nya, globala variablerna som kommer behövas. Det vi ska arbeta med först är ost och därför kommer vi att skapa de variablerna först.

```
#cheese variabel
cheeseeaten=0
cheeseamount=10
currentcheeses=0
cheeseplaces=[[3,3],[2,1],[4,5],[7,3],[2,8],[9,1],[5,7],[8,3],[4,1],[2,7],[5,4
]]
speed=0.2
```

De nya variablerna vi lägger till är cheeseamount, som håller koll på hur mycket ost som ska tillåtas på planen. Currenctcheeses, den har koll på hur många ostbitar som är utplacerade på planen. Den sista ostrelaterade är cheeseplaces som är en array med kordinater där osten kan placeras ut, detta för att ge musens långtidsminne en lättare funktion av att kunna gissa sig till vart osten finns. Den sista variabeln vi lägger till är speed, detta för att kunna ändra hur snabbt musen kommer röra sig.

2) Ost!

Nu ska vi skapa autogenerade ostar, för varje ost som äts upp ska en ny poppa upp. Vi vill innan ta bort den/de manuellt utplacerade ostarna från koden. Försök med detta själv.

```
def Createcheese():
    global currentcheeses
    if cheeseamount>currentcheeses:
        ptpc= randint(0, len(cheeseplaces)-1) #placetoplacecheese
        x=cheeseplaces[ptpc][0]
        y=cheeseplaces[ptpc][1]
        C.create_rectangle(getcoord(x,y), fill="yellow")
        currentcheeses+=1
        global Map
        Map[x][y]=1
```

Ovan är vår nya ostkod, vi börjar med att kalla på den globala variabeln currenctcheeses. Vi kollar om currectcheeses är färre än cheeseamount, om detta är fallet ska vi sätta ut en till ost. Ptpc får ett randomvärde från 0 till cheeseplaces och tar sedan x och y värdet från den platsen i cheeseplaces arrayen. Med detta värde skapas en ny gul rektangel på den angivna platsen och currentcheeses ökar med 1. Det sista vi gör är att vi ändrar i Map, alltså våran karta så dessa kordinater innehåller värdet 1 så programmet vet att det är ost där.

Vi går nu in i Mousemove och lägger till ett anropp till Createcheese precis innan C.update så vet vi att den kollar varje rörelse om en ny ost behöver skapas. När vi ändå är inne i Mousemove vill vi uppdatera:

```
        time.sleep(1) ->      time.sleep(speed)
```

För att ta bort ostar vi hittar så ska vi uppdatera våran befintliga FoundCheese

```
def FoundCheese(mousepos):
    global Map
    if Map[mousepos[0]][mousepos[1]]==1:
        outputtext= "Found Cheese at"+str(mouse[0])+" X  "+str(mouse[1])+ "
Y/nAdded to longterm memory"
        global cheeseeaten
        cheeseeaten+=1
        text.insert(INSERT, outputtext)
        global långtidsminne
        långtidsminne.append([mousepos[0],mousepos[1]])
        C.create_rectangle(getcoord(mousepos[0],mousepos[1]), fill="orange")
        Map[mousepos[0]][mousepos[1]]=0
        global energi
        energi+=10
        global currentcheeses
        currentcheeses-=1
```

Från förra versionen och till denna så är de första raderna desamma med undantaget att vi gör Map global, för att vi senare ska ändra Mapvärden och att vi förlänger vår outputtext.

Vi anropar långtidsminnet och lägger till musens nuvarande position (ostens position). Vi skapar sedan en orangerektangel så vi vet att osten är äten (man kan även göra den grå/vit igen om man vill det). Vi sätter sedan Mapens kordinater till 0, alltså att det inte längre ligger en ost här.

Vi anropar energi och adderar 10 värden och tar bort en ost från currentcheeses.

3) Energiförbrukning

I energiförbrukning så ska lägga till lite till våran whileloop i botten samt flytta in den i en Moveplanner whileloop för att skapa lite mer dynamiska rörelsemönster och att den håller på tills energin är slut.

```
while energi>0:
    energiförbrukning=randint(1,energi)
    mousemove=MovePlanner(energiförbrukning)
    energi-=energiförbrukning
    while len(mousemove)>0: #Gamla loopen
        steg=[0,0]
        # C.create_rectangle(getcoord(mouse[0],mouse[1]), fill="blue")
        if mousemove[0]=="upp":
                mouse[1]= mouse[1]-1
                steg[1]-=1
        elif mousemove[0]=="ner":
                mouse[1]= mouse[1]+1
                steg[1]+=1
        elif mousemove[0]=="vänster":
                steg[0]-=1
                mouse[0]= mouse[0]-1
```

```
        elif mousemove[0]=="höger":
                steg[0]+=1
                mouse[0]= mouse[0]+1
        Movemouse(steg,Mouse)
        FoundCheese(mouse)
        mousemove.pop(0)
    text.delete("insert linestart", "insert lineend")
    text.insert(END, str(energiförbrukning)+" steps taken, "+str(energi)+"
energy left!")
```

Det vi gör är att vi skapar en loop som säger att så länge energi är mer än 0 så ska vi skapa en energiförbrukning som bestämmer hur mycket många steg vi ska ta i den här vändan. Det kan vara allt mellan ett och den totala energin vi har just nu. Detta kan man ändra till det man själv tycker är rimligt. Vi tar såklart bort den mängden energi vi använder från energi variabeln och efter det får vi gå ner till längst ner i loopen och lägger till så en text skriver ut hur mycket steg vi tagit och hur mycket energi vi har kvar.

4) Rörelse

I sista och viktigaste delen ska vi göra om MovePlanner så vår mus rör sig mycket smidigare, detta går såklart fortfarande att förbättra. Vi ska göra om vår MovePlanner helt så den istället för att bara ge random värden faktiskt ger värden baserade på vart musen har hittat ost, vart han nyligen har gått eller random.

```
def MovePlanner(steps):
    whattodo= randint(0,3)
    if whattodo==0:
        print("kortis")
        korttidsminne= Lookinlongterm(steps)
    elif whattodo==1:
        print("långis")
        korttidsminne= Lookinshortterm(steps)
    else:
        print("random")
        korttidsminne= Wanderrandom(steps)
    print(korttidsminne)
    return korttidsminne
```

Börja med att byta namn på den nuvarande Moveplanner till Wanderrandom, detta är en av de tre möjliga rörelserna.

Det vi gör i nya Moveplanner är att vi skapar ett randomvärde mellan 0-3, om det är 0 så ska den köra metoden Lookinglongterm. Är värdet 1 ska vi köra Lookingshortterm och är den något annat så körs Wanderrandom. Alla värdens resultat sparas i kortidsminnet och returnernas senare, varför det sparas i kortidminnet är för att det ska användas nästa varv för att visa åt vilket håll den gick mest. Beroende på vilken metod som körs så skrivs det ut med värdena i terminalen så man kan se att alla körs och att det blir rätt.

(1) Lookinlongterm

För funktionen Lookinglongterm() så kollar vi i långtidsminnet efter någon gammal ost-position. Hittar vi inte det så körs Wanderrandom(). Hittar vi så tar vi reda på avståendet mellan målet(osten) och musens nuvarande position. Vi tar reda på om vi måste gå upp,ner vänster eller höger och så utformar vi vår steglistas efter de kraven. När vi har lagt in alla steg vi behöver så går han random efter det.

```
def Lookinlongterm(steps):
    if len(långtidsminne)==0:
        return Wanderrandom(steps)
    else:
        Memoryplace= randint(0,len(långtidsminne)-1)
        Minnet=långtidsminne[Memoryplace]
        målX=Minnet[0]-mouse[0]
        målY=Minnet[1]-mouse[1]
        steglista=[]
        while steps>0:
            if målX<0:
                steglista.append("upp")
                målX+=1
            elif målX>0:
                steglista.append("ner")
                målX-=1
            elif målY<0:
                steglista.append("vänster")
                målY+=1
            elif målY>0:
                steglista.append("höger")
                målY-=1
            else:
                steglista.append(Wanderrandom(1)[0])
            steps-=1
        return steglista
```

(2) Lookingshortterm

I shortterm så kollar vi gamla korttidsminnet, om det inte finns något så kör den Wanderrandom(). Annars så skapar vi en array med 4 värden, dessa symboliserar upp,ner,vänster och höger. Det hållet den gick mest åt senast i både lodrätt och vågrätt är hållet den även kommer gå åt denna omgång. Då har jag lagt att den går varannan lodrätt och varannan vågrätt men det kan man ändra till random eller något annat om man vill.

```
def Lookinshortterm(steps):
    if len(korttidsminne)==0:
        return Wanderrandom(steps)
    else:
        meströrelse=[0,0,0,0]
        for minnen in korttidsminne:
            if minnen=="upp":
                meströrelse[0]+=1
            elif minnen=="ner":
                meströrelse[1]+=1
            elif minnen=="väster":
                meströrelse[2]+=1
            elif minnen=="höger":
                meströrelse[3]+=1
        Lodrätt=""
        Vågrätt=""
        if meströrelse[0]>meströrelse[1]:
            Lodrätt="upp"
        else:
            Lodrätt="ner"
        if meströrelse[2]>meströrelse[3]:
            Vågrätt="väster"
        else:
```

```
        Vågrätt="höger"
    steglista=[]
    while steps>0:
        if steps%2==0:
            steglista.append(Lodrätt)
        else:
            steglista.append(Vågrätt)
        steps-=1
    return steglista
```