# Labyrinten

## Ett project under utbildning I python programmering för Maskin-inlärning

### Av Niclas Flach

*https://github.com/niclasflach/ML-Labyrinth*



## Mål med projektet

Mitt initiala mål var att bygga en styrning för Labyrint-spelet som kunde få kulan att ta sig från starten till slutet. Detta skulle naturligtvis visa sig vara svårare än jag först trodde.

Ambitionen var alltså att bygga en Reinforcement learning modell som fångar bild vi en kamera som är kopplad till datorn som styr spelet för att klara uppgiften.

## Kuriosa

Jag tycker det är fantastiskt roligt att bygga grejjor och se det fungera i verkligheten. Det är detta som ledde mig till detta projektet och jag insåg från början att en stor del av det inte har med Maskin-Inlärning att göra. Men det är likväl lärorikt och har givit mig många insikter. För att det inte skulle växa till orimliga proportioner så valde jag att bygga så mycket som möjligt med enkla legobitar och lite smältlim.

## Förutsättningar

En första förutsättning var naturligtvis att ha en kamera som passar för uppgiften.

En passande styrning för att kontrollera lutningen på Labyrint spelet

Ett script som tolkar bilden och innehåller de funktioner som krävs för att kontrollera styrningen

Belysning för att kameran skall ha bästa förutsättningar att ta bilder som blir lika under de flesta förutsättningarna.

Ett Labyrint spel naturligtvis

## Exekutiv sammanställning

Jag har skapat ett skript som via en arduino kontrollerar tre stycken servon. Ett av dessa servon används för att återföra kulan till starten i de fall den ramlar ner i ett hål. De andra två kontrollerar lutningen på brädet. En kamera som fångar bilder med hjälp av Opencv och behandlar bilden så den är lämplig att hanteras av RL-modellen. Jag har under en tid tränat modellen för att se om jag kan uppfatta att den blir bättre. Jag tycker mig en smula optimistiskt att jag kan se att den är bättre

Jag har under projektet tydligt visat att den lär sig under vägen. Modellen blir bättre och bättre på att hantera spelet. Det har visat sig vara en större utmaning än jag väntade mig att formulera och koda belöningar och bestraffningar. Vi minsta obalans så lär sig modellen att utnyttja systemet för att få så mycket belöning som möjligt på kort tid, exempelvis genom att ta några snabba poäng köra ner kula i ett hål för att snabbt ta samma belöning en gång till. Detta kan till viss del avhjälpas genom att ändra Gamma värdet för att få större utväxling av framtida belöningar.

Något som jag skulle och kommer vilja utforska vidare är Learning rate. Antal Epochs i varje batch och studera hur jag kan förbättra input dvs observation space för modellen.

Jag har läst lite om hur man kan förträna modellen vilket också verkar intressant. Man skulle kunna träna modellen i förväg på en hel platta utan hål för att den skulle lära sig hantera lutningen och styrningen av kulan.

Jag har börjat titta på hur man kan förbättra loggningen med Callbacks för att mer detaljera studera inlärningen och därigenom kunna upptäcka när något inte går som förväntat.

## Utveckling steg för steg

För att få en bild av hur man kan arbeta med Reinforcement learning började jag studera några exempel som man kan finna på Youtube. Som vanligt finns det ett överflöd av människor som givit sig på att förklara detta på ett pedagogiskt vis, tyvärr misslyckas dom flesta. Jag bestämde mig för att göra ett enkelt exempel som finns tydligt dokumenterat.

Stablebaselines3

Biblioteket som används i exemplet är stable-baselines3.

Det var mycket lärorikt, och gav mig en förståelse och insikt i hur jag kan göra.

(ref: <https://github.com/araffin/rl-tutorial-jnrr19/tree/sb3> )

Stable baselines 3 är en uppsättning med Reinforcement learning algoritmer för eller i PyTorch.

Man kan under träning använda sig av Tensorboard för att följa inlärningen för att säkerställa att den gör framsteg och att inlärningen inte går åt fel håll eller att den “över-passar(over fitting)” modellen.

### Miljön

För att kunna träna modellen så skapas en miljö med några egenskaper och metoder

Dessa metode och egenskapera används sedan av modellen under träning

Man skapar en Class som ärver av Env i gymnasium enligt nedan:

from gymnasium import Env

class LabyrintGame(Env):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.observation\_space = Dict(

{

"picture": Box(low=0, high=255, shape=(200, 200, 3), dtype=np.uint8),  
 "position\_x": Box(low=0, high=255, shape=(1,), dtype=np.uint8),  
 "position\_y": Box(low=0, high=255, shape=(1,), dtype=np.uint8),  
 "visited": Box(low=0, high=255, shape=(200, 200, 1), dtype=np.uint8),  
 "servoPosition1": Box(low=0, high=32, shape=(1,), dtype=np.uint8),  
 "servoPosition2": Box(low=0, high=32, shape=(1,), dtype=np.uint8),  
 "distanceToPoi": Box(low=0, high=232, shape=(1,), dtype=np.uint8),  
 "nextPoi": Box(low=0, high=232, shape=(2,), dtype=np.uint8),  
 }  
 )

Klassen LabyrintGame har en dunder init som definierar det vi kallar för observation space. Detta är den input modellen får under träning och det är endast det den “ser”.

Detta har jag byggt på under tiden och det är långt ifrån fullständigt. Observation\_space är en Dictionary som har åtta enskilda observationer i sig. **Picture** är en bild över brädet nerskala men i fullfärg. **position\_x** och **position\_y.** Detta får jag tackvare att jag målat kulan grön(lättast att separera) och sedan hittar jag positionen för kkulan genom HoughCircles-funktionen i Opencv

lower\_red = np.array([56, 133, 24])  
upper\_red = np.array([84, 255, 121])  
mask = cv2.inRange(hsv, lower\_red, upper\_red)  
ball\_position = cv2.HoughCircles(red\_ball,cv2.HOUGH\_GRADIENT,1.2,100,param1=100,param2=9,minRadius=2,maxRadius=100,)

**Visited** är helt enkelt en representation av var kulan varit under denna omgång.

**ServoPosition1 & 2** är vilken lutning brädet har i de två axlarna för tillfället

**DistanceToPoi** och **nextPoi** är det senaste som är tillagt och det är för att guida kulan genom labyrinten rätt väg. Detta har jag lagt till för att jag upptäckte att modellen började fuska genom att helt enkelt med hög fart försöka passera över hålen, vilket var mer av en chansning men lyckades tillräckligt ofta. Jag har helt enkelt skapat en lista över “Waypoints” som den skall passera. Lite som i masken implemenationen vi gjort under kursen så får modellen en belöning vid varje punkt i listan. DistanceToPoi är helt enkelt Distance to point of interest alltså avstånd till nästa punkt där en belöning erhålls. NextPoi är koordinaten där belöningen erhålls.

I klassen finns en funktion som heter get\_observation. Det är helt enkelt denna metod som samlar in observationerna och returnerar dom. Denna metod anropas vid varje timestep av step metoden som utför de actions som modellen ber om. Jag har valt att ha 5 diskreta möjliga actions. Luta brädet “upp”, “ner”,”höger”,”vänster” samt “no action”. Om det sista väljs så görs helt enkelt inget annat än att modellen får en observation. I övrigt så räknar step-metoden fram belöningen(eller bestraffningen) som skall tilldelas. Step metoden har skarpa krav på vad den skall returnera då den anropas av modellen.

Observation: Har vi berört ovan, helt enkelt så returneras den Dictionary som beskrivits

Reward: en INT med ett värde. Plus är belöning och minus bestraffning.

Done, är en BOOL, om den är 1(True) så har spelet misslyckats och måste börjas om(genom att reset metoden anropas)

Truncated:

Info:

### Reset metoden

Reset metoden gör inget mer än att nollställa vissa observationer och se till att kulan återförs till start-positionen. Den körs ända till dess att kameran ser kula inom ett definierat område(startpositionen)

Då jag upptäckt att inlärning tar en lång tid har jag gjort så att jag regelbundet sparar modellen och när jag startar upp så laddas den gamla modellen in. Det går inte att ladda in modellen om man ändrat någon eller några parametrar i exempelvis observation\_space vilket kanske kan tyckas vara naturligt.

För att underlätta felsökning har jag också en Debuggning variable som kan sättas till True. När den är satt till True görs ingen inlärning men alla annan funktionalitet körs som vanligt. Till detta så visas fångade bilder och parametrar på skärmen.

cam = cv2.VideoCapture(0)  
arduino = serial.Serial(port=SERIAL\_PORT, baudrate=9600, timeout=0.1)  
env = LabyrintGame()CHECKPOINT\_DIR = "./train/"  
LOG\_DIR = "./logs/"  
env.reset()

debuggin = False

if debuggin:  
 while True:  
 obs, red\_ball, new\_pos = env.get\_observation()  
 if env.get\_done():  
 print("spel slut...")  
 time.sleep(3)  
 env.reset()

if new\_pos:  
 print("ny position")

# if frame is read correctly ret is True  
 # show pictures for testing  
 cv2.imshow("obs", obs["visited"])  
 cv2.imshow("red", red\_ball)  
 if cv2.waitKey(1) == ord("q"):  
 break

else:  
 model\_path = "./model\_med\_poi.zip"

if os.path.exists(model\_path):  
 model = PPO.load(model\_path, env)

else:

model = PPO(  
 "MultiInputPolicy",  
 env,  
 batch\_size=10000,  
 tensorboard\_log=LOG\_DIR,  
 verbose=1,  
 n\_steps=10000,  
 learning\_rate=0.009,  
 )  
 for i in range(300):  
 model.learn(total\_timesteps=20000, progress\_bar=True)  
 print("saving model....")  
 model.save(model\_path)