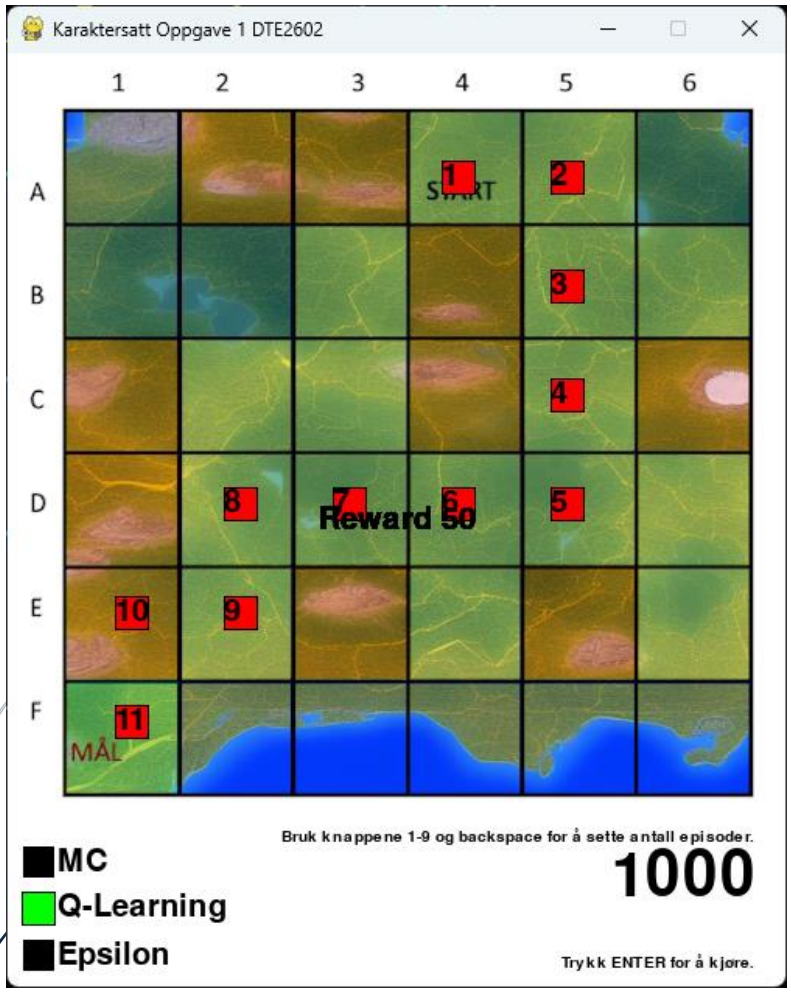
13.10.2024

Q-Learning

Karaktersatt oppgave 1

DTE2602 – Introduksjon Maskinlæring og AI



Eirik Tennøfjord

STUDENT

Innhold

1. Terminologi

Agent	En agent er selve koden. Eller den som tar beslutninger. Denne bruker informasjon fra miljøet for å utføre handlinger for å oppnå et mål.*[3]*
Miljø	Miljøet er selve omgivelsene agenten opererer i. Dette kan være et fysisk miljø, eller en simulert verden. Miljøet gir tilbakemelding til agenten i form av belønning/straff og tilstand. *[3]*
Episoder (Epochs)	En episode er en hel sekvens fra start tilstand, til slutt tilstand. For å trene agenten er det nødvendig å ha flere episoder, slik at man gradvis kan forbedre seg.*[3]*
Reinforcement Learning (RL)	Er en type maskinlæring hvor man benytter agenter som lærer av tidligere handlinger i et miljø. *[3]*
Belønning	For hver handling vil agenten få en belønning eller en straff. Dersom handlingen er ønskelig kan man få belønning, og dersom handlingen ikke er ønskelig kan man få en straff. *[3]*
Q-Læring	Q-læring er en type RL, som brukes for å lære agenter til å ta kalkulerte beslutninger i et gitt miljø. *[2]*
Episoder/Epochs	Vil si en syklus, altså én episode tilsvarer stegene fra start til slutt. Når man jobber med RL, bruker man ofte flere episoder for å lære systemet. *[2]*
Q-Matrise	Er en tabell som lagrer «kvaliteten» til de forskjellige handlingene i de gitte tilstandene. Brukes for å finne de gunstige stiene. *[2]*
Policy	En plan som styrer hvordan agenten skal gjøre beslutninger. *[2]*

2. Introduksjon

I dagens samfunn har utviklingen innenfor kunstig intelligens (KI) skutt fart. Eksempler på dette kan være chat bot, selvkjørende biler, selvkjørende ferger/båter og sorteringsroboter. Flere av disse punktene dreier seg om navigasjon, som er en sentral del av KI. Navigasjon er en av de grunnleggende utfordringene innenfor robotikk og maskinlæring. Det å kunne navigere i et miljø, for deretter å identifisere hindringer og finne den mest effektive veien kan være avgjørende for flere felt.

En viktig del av det å kunne navigere i et miljø, er forsterkningslæring. Der vi bruker en kode (agenten) til å navigere og ta beslutninger basert på miljøet. Q-learning og Monte Carlo er to kjente metoder innenfor forsterkningslæring. Begge disse metodene lærer ved å samle opp belønning over tid. Q-learning lærer ved å kontinuerlig oppdatere en Q-matrise etter hvert som miljøet blir utforsket. Monte Carlo summerer opp belønning og sammenligner disse for å finne den mest effektive ruten. *[2]*

En annen strategi for å navigere er en metode vi kaller epsilon-greedy. Denne kombinerer q-learning, og monte carlo. Ved å benytte denne metoden, så vil man bruke kjent informasjon om hvilke ruter som gir høyest belønning, samtidig som man vil kunne oppdage potensielt bedre ruter, ved å gjøre tilfeldige handlinger.

For å studere disse forskjellige læringsmetodene er et 2D-kart et egnet miljø. Gjennom bruk av belønningsmatriser og simuleringer kan agenten lære seg å finne effektive ruter fra start punkt til slutt punkt, samtidig som agenten kan manøvrere seg rundt hindringer.

Denne rapporten vil undersøke hvordan man med ulike navigasjonsstrategier, som Monte Carlo og Q-learning kan finne de mest effektive rutene i et 2D-kart. Formålet er å kunne vurdere hvordan de ulike metodene fungerer, samt hvor effektive de er. Det er også ønskelig å se på antall episoder (epochs), og hvordan dette spiller inn på de forskjellige strategiene. I tillegg til dette skal det også være forskjellig type landskap som fjell og vann, noe som kan skape problemer når man skal bevege seg innenfor disse områdene. Vi tar utgangspunkt i at det er en robot som skal utforske ukjent terreng.

Hypotesen i denne rapporten er at en kombinasjon av disse metodene kan gi en optimal løsning for robotens navigasjon. Q-learning vil være mest effektiv hvor man har et stort antall episoder, mens epsilon-greedy kan gi bedre resultater med et mindre antall episoder. Gjennom simulering vil rapporten vise hvordan roboten lærer og forbedrer sine valg over tid, og hvordan den finner de mest effektive veiene fra start til slutt.

3. Teori

3.1 - Monte Carlo Simulering

Monte Carlo er en kjent metode som benytter seg av tilfeldigheter, og derfor oppkalt etter det kjente kasinoet Monte Carlo i Monaco. I denne metoden blir alle handlinger som skal gjøres bestemt av tilfeldigheter. [4]

For hver episode som blir gjennomført vil man sammenlikne oppnådd belønning med den daværende beste belønningen fra tidligere episoder. Dersom denne den nye belønningen er høyere enn de tidligere, vil denne bli brukt som «beste» løsning.

I sin enkleste form (som er benyttet i denne rapporten) så krever Monte Carlo prinsippet svært mange simuleringer for å oppnå tilfredsstillende resultater. Andre typer «reinforced learning» benytter seg av å ta mer kalkulerte valg.

3.2 - Markov Decision Process (MDP)

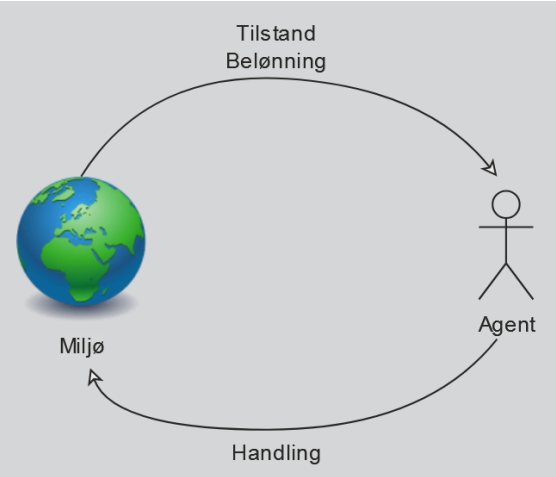
Markov Decision Process vil si at den fremtidige tilstanden kun er avhengig av den nåværende tilstanden.

«Verdien tile en tilstand er summen av alle fremtidige belønninger som kan oppnås fra tilstanden vi er i akkurat nå.» *[2]*

Dette vil da si at man ikke skal se tilbake på de tidligere tilstandene, men fremover mot en fremtidig belønning. For å få til dette bruker man en «discount factor» (γ / gamma).

$$\gamma \in [0, 1]$$

Gamma skal være et tall mellom 0 og 1, og ved å justere denne kan man redusere betydningen av fremtidige belønninger.



Figur 3.2.1 – Q-læring

Dette er sentralt i det vi kaller q-læring hvor en agent utfører en handling i et miljø og deretter får tilbakemelding i form av ny tilstand og belønning/straff. Når man mottar belønning/straff vil man oppdatere q-lærings matrisen sin ved hjelp av likninger som Bellman.

3.3 - Bellman likningen

Bellman-likningen er en matematisk likning som beregner verdien av en tilstand og verdien av de påfølgende tilstandene. Denne er oppkalt etter Richard Bellman som var mannen bak den. Ved å bruke denne likningen kan man dele opp problematikken med belønning i et læringssystem.

Q-learning bygger på Bellman-funksjonene:

$$V(s) \leftarrow \sum_{a} \pi(\alpha|s) \sum_{s^1,r}^{1} p(s', r|s, \alpha)[r + \gamma V(s')]$$

Ligning for Bellman sin verdifunksjon

V(s)	Er verdien av tilstand s
a	Handlinger (action)
s	Tilstand (state)
π	Policy
s’	Neste tilstand
r	Belønning (reward)
p	Sannsynligheten
γ	Gamma (Discount factor)

[2]

Når vi skal beregne verdiene vi setter inn i q-lærings matrisen (Q-Matrisen), bruker vi Bellman likningen for å kalkulere kvaliteten (Q – Quality). Dette gjøres med formelen:

$$Q(s', a) \leftarrow (1 - \alpha) * Q(s, \alpha) + \alpha * (r + \gamma * \max_{a'} Q(s', a'))$$

Q(s’, a) → Den forventede kvaliteten til en tilstand vi ønsker å gå til gitt en handling a
$(1 - \alpha) * Q(s, \alpha)$ → Gammel kunnskap fra forrige tilstand.
$\alpha * (r + \gamma * \max_{a'} Q(s', a'))$ → Ny kunnskap fra nåværende tilstand og nåværende action.

Ved å benytte seg av stor alpha tar man mer hensyn til den nye belønningen, og ved å benytte stor gamma tar man mer hensyn til fremtidig belønning. *[3]*

3.4 – Droner for levering

Et av satsingsområdene til Amazon er varelevering ved hjelp av droner. Noe som nylig har blitt godkjent å teste ut i Storbritannia [5]. Dette kan føre til at man har et mindre behov for biler for levering av pakker, noe som igjen kan være positivt med tanke på klima [6]. Det er også ikke bare Amazon som har sånne prosjekter. I Rwanda leverer firmaet «Zipline» medisinsk utstyr rundt i landet ved hjelp av leveringsdroner. *[7]*

Dette kan by på flere forskjellige problemer med tanke på hindringer i form av trær og bygninger. Samtidig er det viktig at disse systemene overholder tidsfristene for levering som er satt. Spesielt for «Zipline» kan det være kritisk å kunne levere innenfor tidsfristen, siden det er snakk om medisinsk utstyr.

Andre ting som kan gjøre leveringene problematiske er miljøet (vær og vind). Dette vil føre til at vi har et dynamisk miljø, som kan endre seg underveis. Noe som dronene/agentene må ta hensyn til når de skal levere.

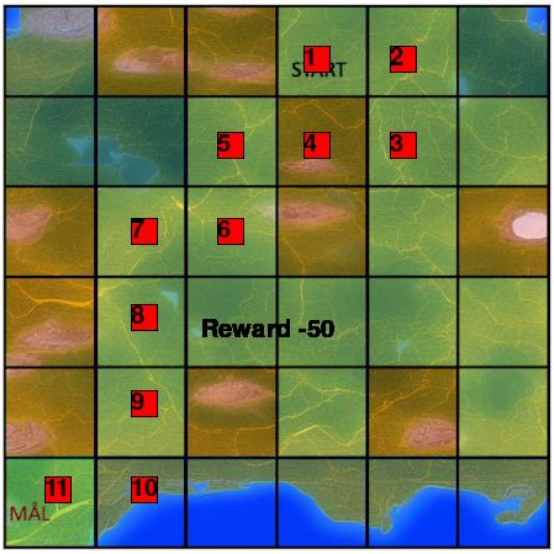
For å håndtere disse utfordringene kan man bruke navigasjonsstrategier som Q-Learning og Monte Carlo metoder. Ved å bruke disse metodene kan man lære dronene de optimale rutene basert på tidligere leveranser. Q-læringen kan bidra til å utvikle Q-matrise som kan lagre verdier som tilstand og handlinger, slik at dronene kan ta kalkulerte beslutninger.

4. Metode

For å utvikle agenten til å navigere i 2D-kartet, ble de benyttet forskjellige metoder innenfor forsterkningslæring. Dette inkluderte: Monte Carlo, Q-learning og Epsilon-Greedy policy. Selve arbeidet ble gjennomført i Python, og ved hjelp av visualiseringsverktøyet "PyGame".

Figur 4.1: Her kan man velge hvilken metode som skal benyttes.

I denne rapporten er det laget klart for tre forskjellige typer maskinlæringsmetoder. Disse kan man velge ved hjelp av «radiobuttons» i programmet (figur 3.1). Her kan man også skrive inn hvor mange episoder som skal benyttes for de forskjellige metodene (nummer tastene 0-9 og backspace).

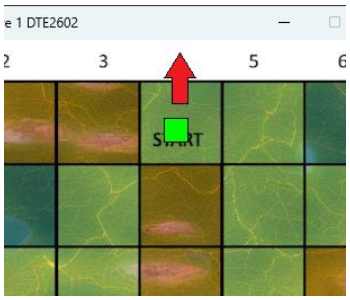


Figur 4.2: Visning av beste sti

Når koden er ferdig kjørt, så vil man få opp i GUI hvilken sti programmet har kommet frem til er den beste.

4.1 - Monte Carlo

Monte Carlo ble iverksatt ved at man setter agenten/roboten i start posisjon, deretter fikk den gjøre tilfeldige handlinger som i dette tilfellet var opp, ned, venstre og høyre. Hver gang man kommer til en ny tilstand vil man oppdatere episodens totale belønning.



Figur 4.1.1: Agent gjør en handling som vil føre man utenfor kartet.

I Monte Carlo, kan man også gjøre valg som fører til at man kan forsøke å gå gjennom ytterkanten/veggen på kartet (som vist i figur 4.1.1). Denne gir ifølge belønningsmatrisen en negativ belønning. I denne rapporten er det valgt å ikke summere opp straffen fra å prøve å gå utenfor kartet (for MC), siden agenten uansett ender i samme tilstand.

Etter hver episode vil man sjekke om den gjeldende episoden har høyere belønning enn de tidligere episodene. Dersom den er høyere, vil man lagre denne som foreløpig beste sti. Når programmet initialiseres blir den beste løsningen satt til « -inf» og ikke 0. Dette er fordi at flere av resultatene kan gi negative verdier, så ved å sette denne så lav vil alt som kommer etter være større.

4.2. Q-Learning/Greedy

Q-Læringen ble implementert ved å initialere en Q-matrise som holdt styr på de forskjellige belønningene for hver tilstand og handling. Agenten startet fra en tilfeldig posisjon i kartet og utfører en handling basert på valgt policy (π). I denne funksjonen ble det gjort tilfeldige valg.

For hvert steg vil belønningsmatrisen bli oppdatert i henhold til Bellman ligningen. Etter at man har kjørt alle episodene, så vil man kalle funksjon for å finne sti basert på verdiene i q-matrisen. For å finne beste sti basert på q-matrisen har det blitt iverksatt en greedy funksjon, som i en gitt tilstand skal velge den handlingen som gir høyest mulig belønning.

4.3. Epsilon-Greedy

Denne metoden bygger på de samme prinsippene som q-læring og Monte Carlo, og kombinerer disse. Ved hjelp av en parameter satt i robot filen (epsilon), vil man kunne justere på hvor ofte man skal gjøre en tilfeldig handling, i stedet for å gå etter den som gir mest belønning. Dersom man har en høy verdi på epsilon (alltid mellom 0 og 1), så vil man oftere gjøre tilfeldige handlinger. Dersom man ikke gjør en tilfeldig handling, vil man velge den handlingen som gir best belønning i henhold til Q-matrisen.

4.4. Diverse

For greedy explotation funksjonen som blir brukt i både Epsilon og Q-learning i denne rapporten, så er det også gjennomført en funksjon for «visited». Dette er for å forhindre at man går i en uendelig loop mellom to ruter. Dersom alle nabo-tilstandene er besøkt, så vil man velge en tilfeldig.

Visited funksjonen er en matrise med alle tilstandene. Hver av tilstandene har en påfølgende verdi som er True/False. Det er også lagt til et ekstra ledd i hver kant for å lagre veggene som «visited». Disse blir satt til besøkt fra start.

Forsøkene i denne rapporten er utført med følgende verider:

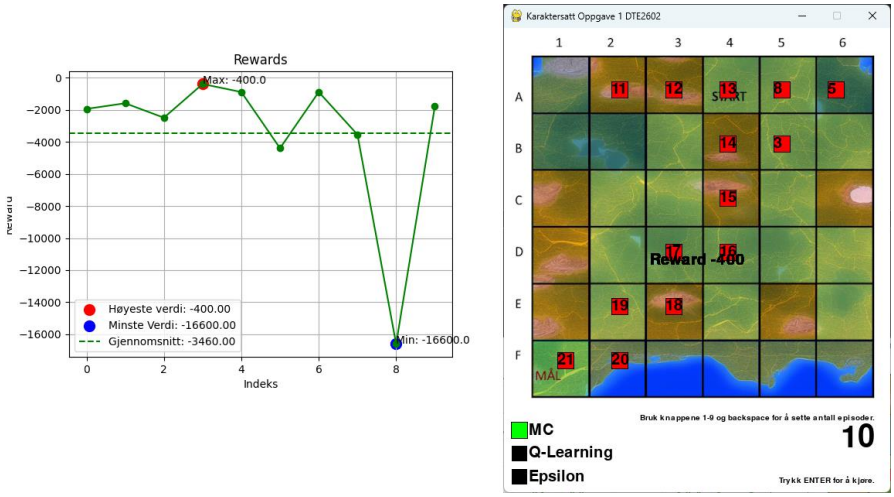
Verdi	Variabel	Forklaring
0.4	Alpha (Læringsrate)	Denne er satt til en lavere verdi siden vi opererer i et statisk miljø. Det er ikke noen store endringer. Dersom det hadde vært et mer dynamisk miljø, ville det vært mer nyttig med en høy læringsrate. [1]
0.9	Gamma (Discount factor)	Denne er satt til en høy verdi, fordi belønningen kommer senere. Når denne er høy, vil man derfor vektlegge fremtidig belønning. Dersom man var usikker på belønningen kunne man brukt en lavere verdi her. [1]
0.1	Epsilon	Denne er satt til denne verdien etter en del prøving og feiling. Dersom man hadde hatt mer tid, skulle man helst hatt iverksatt en funksjon som gradvis justerte ned epsilon variabelen utover i episodene.
0	Hvitt felt	Har valgt å gi denne verdien «0», siden man ikke skal bli straffet for å gå på disse feltene, men samtidig kommer belønningen senere.
-50	Rødt felt	De røde feltene (fjellområdene) har fått verdien «-50». Dette fordi man skal helst unngå å gå på disse feltene, med mindre man må.
-100	Blått felt	De blå feltene har fått verdien «-100». Dette fordi at det fortsatt skal være mulig å gå innom disse feltene, men man bør heller prioritere hvitt eller rødt fremfor blått.
-1000	Vegger	For å unngå at agenten prøver å gå inn i veggene, så har disse fått en høy straff.

5. Resultat

I denne delen av rapporten er det bestemt å vise resultatet av de forskjellige typene læring som er gått gjennom tidligere.

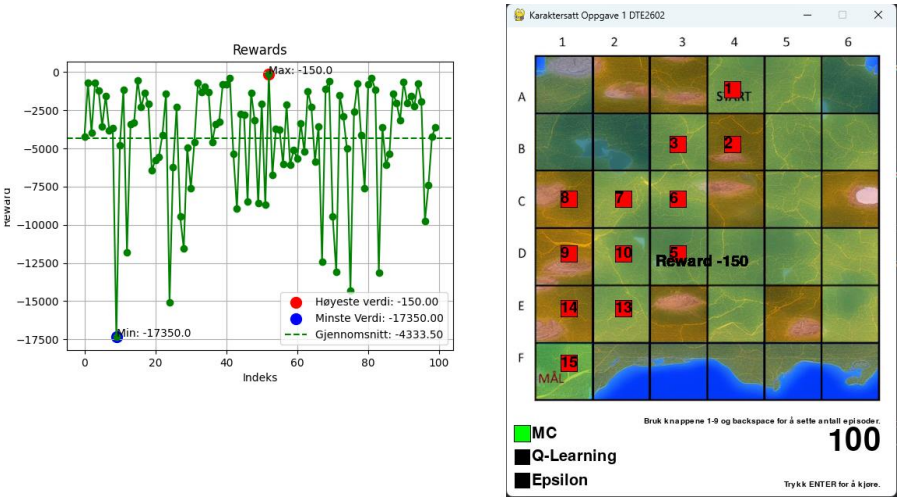
5.1 Monte Carlo

For å komme frem til et resultat for denne metoden har jeg valgt å vise resultatet av simulering kjørt med 10, 100 og 1000 episoder.



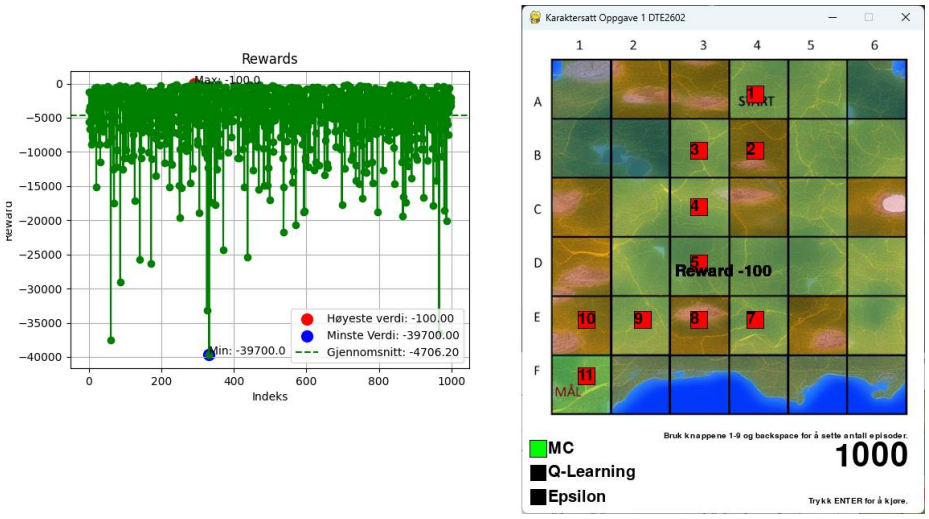
Figur 5.1.1: Monte Carlo med 10 simuleringer

Når det gjelder simuleringen med 10 episoder, så er det svært mye tilfeldigheter knyttet til dette resultatet. I denne spesifikke testen var den beste oppnådde belønningen -400, og den dårligste belønningen var -16,600. Noe som er et svært stort sprik.



Figur 5.1.2: Monte Carlo med 100 simuleringer

Når vi økte antallet episoder til 100, så ser vi at en bedre rute er funnet. Den er ikke så veldig mye bedre enn ruten som ble funnet med 10 episoder. Samtidig ser vi at den dårligste belønningen oppnådd er -17,350. Og gjennomsnittet er -4333.5.

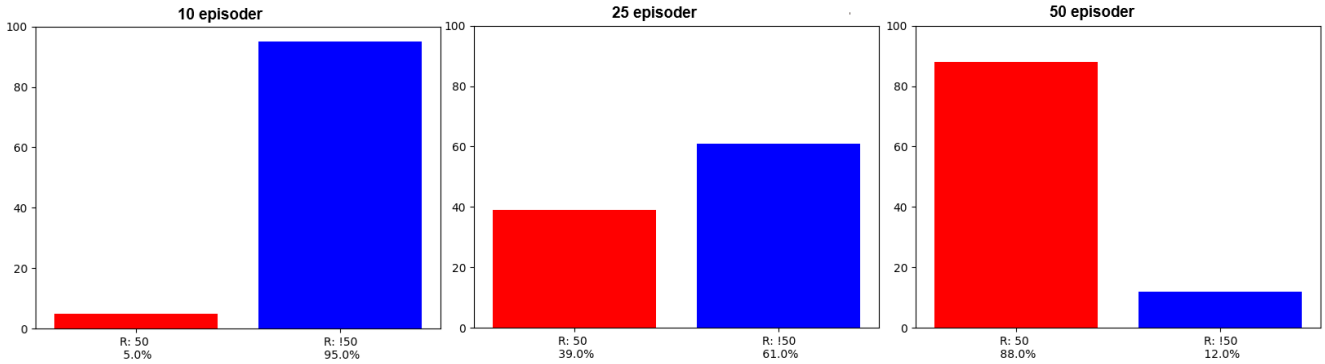


Figur 5.1.3: Monte Carlo med 1000 simuleringer

Til slutt har det blitt gjennomført 1000 episoder. Med denne simuleringen har vi funnet den til nå beste ruten, men det er fortsatt ikke den optimale ruten. Samtidig kan vi bemerke oss at belønningen i dette spesifikke forsøket kun har en forskjell på 50 mellom 100 og 1000 episoder.

5.2 Q-Læring/Greedy

For testing i henhold til q-læring har det blitt bestemt å gjøre testene litt annerledes enn på Monte Carlo, på grunn av deres virkemåte. De følgende testene er gjort med 10, 25 og 50 episoder. For hver av disse er det kjørt 100 simuleringer, for å få grafer vi kan analysere. Her begynner man også å få den optimale ruten, så i stedet for å vise grafer som på Monte Carlo, vil det her bli vist søyle diagram med antall optimal rute og ikke optimal rute (Optimal rute vil si den beste ruten oppnåelig).



Figur 5.2.1: Q-læring med «Greedy route» for 10, 25 og 50 episoder med 100 simuleringer.

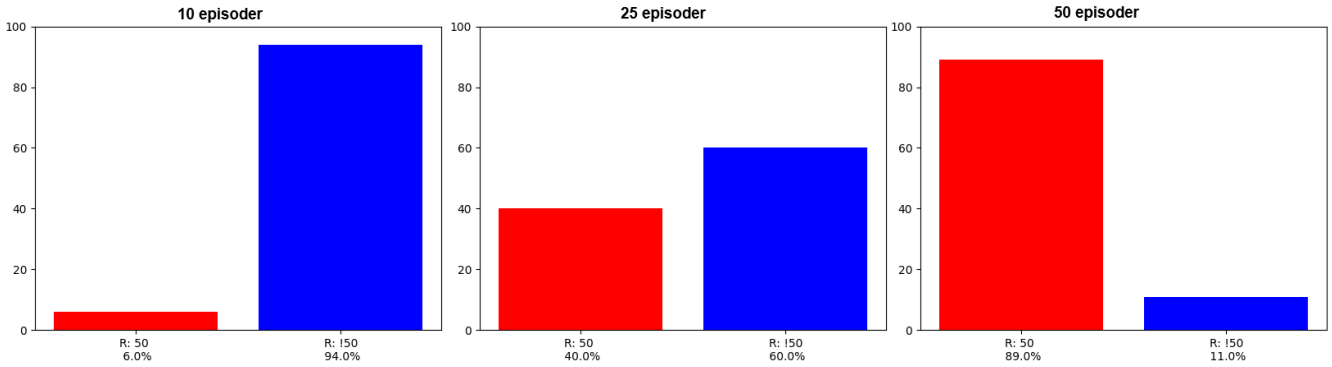
Etter 100 simuleringer med 10 episoder, ser vi at vi begynner å få noen simuleringer som finner den optimale ruten. Men det er fortsatt en god del som er feil.

Når vi øker antallet episoder til 25, ser vi at resultatet begynner å bli mye bedre. I denne spesifikke testen ble det 39% av simuleringene som klarte å finne den optimale stien.

I den siste målingen med 50 episoder, kan vi se at andelen med optimal rute har blitt mye større enn andelen som ikke klarer å finne den. Men ut ifra disse resultatene kan vi se at det fortsatt kan være en del feilmargin.

5.3 Epsilon-Greedy

Den siste metoden er epsilon greedy, denne har det blitt bestemt å teste på samme måte som q-læring/greedy.



Figur 5.3.1: Epsilon med «Greedy route» for 10, 25 og 50 episoder med 100 simuleringer

Resultatet for denne testen er svært lik resultatet i q-learning/greedy. Har var det forventet at epsilon skulle være mer effektiv ved mindre episoder enn q-learning/greedy.

6. Diskusjon og konklusjon

Ut ifra de resultatene som ble oppnådd, så ser man at det stemmer med antagelsene i starten når det gjelder Monte Carlo Simulering. Man må ha et stort antall episoder, for å kunne få tilfredsstillende resultater.

Vi ser også fra resultatene for Q-læring/Greedy at agenten lærer underveis, og ved flere antall episoder blir agenten bedre til å ta beslutninger. Dette var også noe som var forventet fra starten av.

Til slutt når Epsilon/Greedy ble testet, var det forventet at denne skulle være bedre når man hadde få episoder, enn hva som ble oppnådd når man benyttet seg av Q-learning/Greedy. Dette stemte *ikke* med de testene som ble gjennomført for denne rapporten. Noen av grunnene til dette kan være måten dette er implementert på i programmet.

Dersom man har flere verdier i q-matrisen som er den samme (som når man initialiserer den og aller verdiene er 0), så vil man i Epsilon/Greedy funksjonen alltid velge «den første» av de beste valgene.

```
max_reward = max(self.q_matrix[state])
action = self.q_matrix[state].index(max_reward)
```

Figur 6.1 – Hvordan metoden for epsilon-greedy er implementert.
Denne index-funksjonen velger det første elementet med «max_reward»

Denne implementasjonen (fig. 5.1), kan føre til at man i stedet for å velge den beste løsningen, flere ganger velger en ikke optimal løsning i starten. To av måtene dette kunne/burde vært løst på kunne vært:

1. Å velge en tilfeldig av de verdiene i q-matrisen for den gitte tilstanden som har samme verdi.
2. Å innføre en gradvis Epsilon. Det vil si at man har en høy verdi for epsilon i starten, og deretter reduseres denne utover i episodene.

Av de to løsningene hadde det vært ønskelig å teste ut nummer to, siden denne ville passet i flere tilfeller.

En annen ting man i Monte Carlo metoden kunne testet ville vært å implementert en «besøkt» liste. Slik at man kunne tvunget agenten til å ikke tilfeldigvis hoppe mellom to tilstander, eller gå innom samme tilstand flere ganger.

Det er ikke lagt til konvergering i dette programmet. Både for epsilon-greedy og q-læringen, så man må alltid fullføre alle episodene selv om det ikke er noen endringer. Dette ble ikke prioritert i dette tilfellet.

7. Kilder

[1] – UiT Norges Arktiske Universitet (2024, 3. Okt) Hyperparametere og konvergens. Canvas. https://uit.instructure.com/courses/34928/modules/items/1071726

[2] – UiT Norges Arktiske Universitet (2024, 3. Okt) Q-Learning. Canvas. https://uit.instructure.com/courses/34928/modules/items/1026554

[3] – UiT Norges Arktiske Universitet (2024, 3. Okt) Grunnleggende konsepter i RL. Canvas. https://uit.instructure.com/courses/34928/modules/items/1026552

[4] - *Grøn, Øyvind: Monte Carlo-metode i Store norske leksikon på snl.no. Hentet 13. oktober 2024 fra https://snl.no/Monte_Carlo-metode*

[5] –*Viner, Katharine: Clear for takeoff? Amazon gets green light to test-fly delivery drones in uk. The Guardian. Hentet 13. Oktober 2024 fra https://www.theguardian.com/business/article/2024/aug/15/clear-for-takeoff-amazon-gets-green-light-to-test-fly-delivery-drones-in-uk*

[6] – *Rao, Devika: The pros and cons of drone delivery. The Week. Hentet 13. Oktober 2024 fra https://theweek.com/tech/drone-delivery-pros-cons*

[7] – *Baker, Aryn: U.S. startup Zipline has teamed up with the Rwandan government to deliver blood supplies by drone. Time. Hentet 13. Oktober 2024 fra https://time.com/rwanda-drones-zipline/*