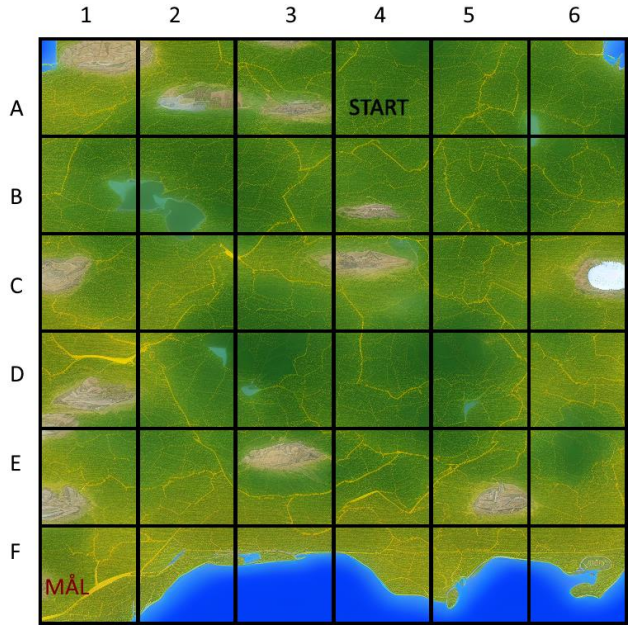
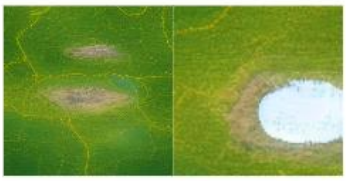
Karaktersatt oppgave 1 | DTE-2602 | H23

I denne oppgaven skal du lage en simulering av en robot som utforsker et ukjent terreng. For enkelhets skyld er terrenget delt opp som et rutenett med 36 tilstander. Roboten skal starte i A4 og skal prøve å finne en trygg vei frem til F1.

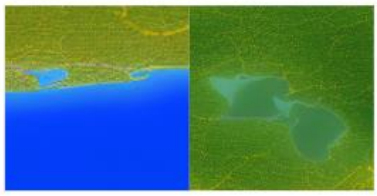


Rundt om i terrenget er det ulike hindringer som roboten må ta hensyn til.

Fjelltopper og bratt, ulendt terreng:

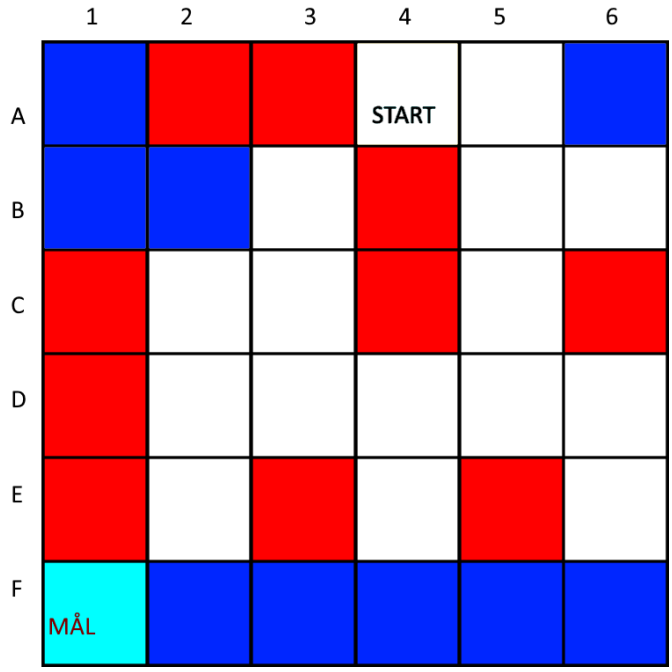


Vann og innsjøer:



Roboten klarer å kjøre i bratt terreng, men det krever mye energi og er forbundet med høyere risiko. Roboten er vanntett, men å kjøre gjennom vann krever enda mer energi, og tar svært lang tid sammenliknet med å kjøre på land eller i bratt terreng.

Basert på dette kan kartet forenkles til følgende figur:



- Røde ruter er fjell og bratt terreng, og mørkeblå ruter er vann. Hvite ruter er forbundet med lav eller ingen risiko.
- Roboten styres med et API som tillater følgende bevegelser: “opp”, “ned”, “høyre” og “venstre”. Hvis roboten står i rute “A4” og gjør bevegelsen “ned” havner den altså i rute “B4” Roboten kan ikke bevege seg diagonalt.

Oppgave

Besvarelsen på oppgaven består av to deler som skal leveres via repository på GitHub Classroom:

- En Python-fil med implementasjon. Teller 60% .
- En rapport i form av et-PDF dokument (f.eks. generert fra MS Word eller LaTeX) med beskrivelse av din egen løsning. Rapporten bør være mellom 3 og 6 sider (A4, skriftstørrelse 11). Rapporten kan skrives på norsk eller engelsk. Teller 40 %

Implementasjonen vurderes etter følgende kriterier:

- **Korrekthet:** Koden svarer på oppgaven og fungerer som den skal.
- **Effektivitet:** Man har gjort gode valg i implementasjonen. Koden trenger ikke være lynrask, men man skal unngå unødvendig tungvinte løsninger.
- **Lesbarhet:** Koden har
    - Beskrivende docstrings for alle funksjoner / metoder
    - En passe mengde kommentarer som tydeliggjør hensikten med koden
    - Beskrivende variabelnavn
    - Formatering som følger anbefalinger gitt i video om kodestandard (trenger ikke alltid følge absolutt alle regler, men bør ha god og «standard» formatering)

Rapporten vurderes etter følgende kriterier:

- **Korrekthet:** Svarer på konkrete punkter etterspurt i oppgaven. Bruker riktig teori og ligninger der dette er relevant.
- **Referanse til kode:** Det skal være en tydelig sammenheng mellom innholdet i rapporten og koden. Vis f.eks. hvordan evt. matematiske ligninger i rapporten er implementert i koden.
- **Lesbarhet og struktur:** Språket er korrekt og tydelig, og skrevet i fullstendige setninger. Dokumentet er tydelig og konsistent formatert, med overskrifter og inndeling som gjør det lett å finne fram i. Selv om oppgaven er gitt punktvis, trenger ikke rapporten «svare» punktvis. Rapporten skal beskrive helheten i løsningen av oppgaven (dvs. Q-learning).
- **Selvrefleksjon:** I rapporten skal du også vurdere ditt eget arbeid. Fungerer løsningen din? Hvis ikke, hva tror du er årsaken? Er det noe du ville ha gjort annerledes?

Oppgaven er delt opp i underoppgaver, med et maksimalt antall poeng oppgitt for hver. Hvis en oppgave ikke er gjort, får man 0 poeng for denne. Man kan få opp til 100 poeng totalt.

1. Implementer klassen «Robot», og definer en R-matrise (reward-matrise) for kartet som er oppgitt. Gjør egne vurderinger angående hvilke verdier som bør brukes i matrisa. Gi en begrunnelse i rapporten for valgene du gjør. **10 poeng.**
2. Implementer metoden get_next_state_mc() som tilfeldig velger en av fire mulige «actions» for ruta roboten står i (opp, ned, høyre, venstre). Dersom roboten står i kanten av rutenettet og prøver å gå ut av kartet, skal den bli stående i samme rute. **5 poeng**
3. Implementer metoden monte_carlo_exploration(), som kjører Monte Carlo-simulering av at roboten beveger seg tilfeldig i kartet (som beskrevet i oppgave 2). Metoden skal ha antall simuleringer som input-argument. Hver simulering starter med roboten i rute A4 og slutter når den når rute F1. Metoden skal beregne total belønning for hver simulering (basert på matrise i oppgave 1), og ta vare på den beste ruta og den høyeste belønningen den har funnet. Kjør simuleringen 100 ganger og beskriv den beste ruta roboten finner (gjerne med en figur). **25 poeng**
4. Definer en Q-matrise for roboten, og implementer metoden q_learning(). Metoden skal la roboten utforske kartet og samtidig oppdatere Q-matrisa underveis. Metoden skal ha antall

episoder som input-argument. For hver episode plasserer du roboten i en tilfeldig starttilstand, som slutter når den når F1 - måltilstanden. Velg en av to policy-varianter (gir maks. 30 poeng):

   a. Enkel variant: Roboten velger 1 av 4 mulige bevegelser tilfeldig (samme som Monte Carlo i oppgave 3) **20 poeng**
   b. Mer kompleks/effektiv variant («epsilon-greedy»): Roboten velger tilfeldig bevegelse en andel av tida (f.eks. 80% av gangene), og velger alternativet som virker best (ut fra Q-matrisa) resten av gangene. Krever alternativ metode for å finne neste tilstand (get_next_state_ge()). Andelen tilfeldige valg ("epsilon") bør være en input-parameter. **30 poeng**

5. Implementer metoden greedy_path() som returnerer beste rute fra start til mål (basert på «trent» Q-matrise), med tilhørende total belønning. **10 poeng**
6. Eksperimenter med å kjøre Q-learning-programmet med ulike antall episoder (og evt. ulike varianter av epsilon-greedy policy), og vis hvor god rute roboten finner for hvert eksperiment. Klarer den å finne den optimale ruta? Det kan hende du må justere reward-matrisa (se oppgave 1) for å få gode resultater. Sammenlign gjerne med resultatene i oppgave 3. **20 poeng**