INFO 180 – Metodar i kunstig intelligens

Oblig-oppgåve 3 – 18.-22.okt 2021 – Null-sum-spel

OBLIGATORISK! Denne oppgåva må gjennomførast og godkjennast for å få gå opp til eksamen.

Innleveringsfrist på Mitt UiB: 25. oktober 16:00

I denne oppgåva skal vi jobbe med eit spesielt spel, og bidra til å lage eit program som spelar dette spelet godt. Spelet er eit såkalla null-sum-spel med to spelarar som har perfekt informasjon og som spelar etter tur.

Spelet er som følgjer og kan for eksempel modellere ein (heller kunstig) situasjon i eit reality-show eller i ei verksemd der prosjektleiarar konkurrerer om status. Men her skal vi ta utgangspunkt i ein situasjon som oppstod mellom to venner (Åge og Linda) som likar å arrangere festar. Dei byrja ein vennskapleg krangel om kven som kunne arrangere den beste festen. Og dette er sjølvsagt avhengig av kven som kan komme. Dei bestemte seg derfor for å konkurrere om å lage den beste festen (som skal skje samtidig). Dei får 11 gjester kvar – dei må velje blant 22 felles venner. Linda ville gjerne arrangere eit Toga-party, medan Åge ville arrangere eit lite Rave-party.

Kvar av dei 22 vennene vil ha forskjellig veremåte knytt til dei to ulike fest-konsepta, så kvar av dei har ulik kompatibilitet til Rave- og Toga-party. Vi modellerer dette som eit tal mellom 1 og 9 der høgt tal tilsvarer høg kompatibilitet til festkonseptet. Både Åge og Linda har all kunnskap om kvar av dei 22 vennene sin kompatibilitet til fest-konsepta. Ideen er at høg sum av kompatibilitet for utvalde fest-deltakarar gjev høg sjanse for suksess i festen. Samtidig er det slik at konkurrentens fest-suksess faktisk påverkar oppfatninga av eigen suksess i konteksten (begge har som mål å lage ein betre fest enn vennen), så målet er óg å redusere konkurrentens kompatibilitet til minst mogleg. Grad av suksess i dette utveljings-spelet er altså å velje ut dei gjestene som maksimerer sjansar for ein god fest, samtidig som det minimerer motstandarens sjansar for god fest. Differansen mellom summen av kompatibilitet for dei 11 utvalde er altså det endeleg utfallet. Spelet er slik at ein vel ein og ein kandidat etter tur.

Dette er ikkje eit trivielt spel, fordi det ofte ikkje lønner seg å velje den kandidaten som er best isolert sett, ein må og sjå på konsekvensen for motstandaren. Eksempel: Ein forenkla versjon med berre to kandidatar har kompatibilitet som i tabellen under:

	Kandidat 1	Kandidat 2
Toga (Linda)	6	7
Rave (Åge)	9	5

I dette tilfellet vil det for Linda lønne seg å velje kandidat 1, slik at Åge endar opp med kandidat 2. Med 22 kandidatar har vi i alt 1.124.000.727.777.610.000.000 spel så speltreet er umuleg å søke gjennom på fornuftig tid.

Til å spele dette spelet vert det lagt ut python-program som de kan bruke:

- python-program som gjennomfører sjølve spelet (SelectParty.py)
- python-linjer som startar og køyrer spelet mellom menneske og maskin (Main.py)
- ei klasse som representerer informasjon om kandidatar som kan veljast (Candidate.py) . Denne klassen lagar og ein ny instans av spelet ved å gje kandidatar kompatibilitet ved hjelp av ein tilfeldig tal-generator. Den er laga slik at summen av kompatibilitetar for alle kandidatane er 100 for begge fest-typane.
- ei klasse som representer informasjon om status for spelet (GameStatus.py), blant anna med kven som er i trekket, kor mange kandidatar som er igjen etc.
- Ei abstrakt klasse Player som representerer informasjon om ein spelar i programmet (Player.py).
- Ein konkret klasse HumanPlayer.py som handterer spelet for ein menneskeleg spelar.
- Ein abstrakt klasse ComputerPlayer.py som handterer spelet for datamaskina. Den vel trekk ved eit minimax-søk gjennomført av ei alphabeta-algoritme. Den implementerer ikkje evalueringsfunksjonen for å vurdere kor god ein utbyttenode er for datamaskina.
- SimpleComputerPlayer.py implementerer ein enkel evalueringsfunksjon som vurderer ein utbyttenode i speltreet til å vere summen av kompatibilitets-scorane til dei valde kandidatane for den som er i trekket (toMove) minus den same summen for den andre spelaren.
- GameTreeNode.py har det som trengs for å representere ein node i speltreet slik at vi kan gjennomføre eit søk. Blant anna er det denne som gjennomfører alphabeta-søket.

Oppgåvene de skal gjere er følgjande:

1. De skal lage ein ny ComputerPlayer-subklasse som har ein annan, litt meir avansert evalueringsfunksjon enn SimpleComputerPlayer og teste ut denne ved å modifisere SelectParty-klassen. Det er mulig å endre MAX-DEPTH (maksimal søkedjupn) i GameTreeNode for å teste ut funksjonaliteten. I utgangspunktet er MAX-DEPTH satt til 4. Hugs at denne evalueringsfunksjonen må vurdere ei stilling ut frå spelaren i trekket (toMove) sitt synspunkt. Slik spelet fungerer kan det for eksempel vere verdt å sjå på kor mykje kompatibilitets-score som er igjen for kvar av spelarane. Det er trulegvis ein fordel å ha meir igjen enn den andre spelaren. GameStatus-klassen har ein funksjon som finn kompatibilitets-scoren som er igjen for ein spelar.
2. De skal lage eit program som køyrer spel mellom to ComputerPlayer-ar, dvs. ingen menneskeleg spelar. Dvs dei skal køyre mot kvarandre 100 gonger. Dette kan gjerast ved å lage ein ny variant av Main.py (for eksempel CompVComp.py), der de set player1 og player2 til kvar sin sort ComputerPlayer, lar spelarane få starte kvar sin gong mens ein summerer opp resultata slik at ein ser kven som spelar best. Kommenter gjerne vekk utskrift slik at du ikkje får all utskrifta undervegs i spela. For å få dette til å gå i rimeleg tid, kan de redusere på MAX-DEPTH. Rapporter resultatet av denne køyringa. Kva for ComputerPlayer-klasse er best gjennom mange spel?

Google Colab:

Python er eit programmeringsspråk som i mange samanhengar ikkje er spesielt effektivt (heller tvert i mot), og dette er ein slik samanheng. Mange av dykk vil kanskje ikkje ha datamaskiner som i rimeleg tid vil kunne spele dette spelet med MAX-DEPTH=4.

Google Colab er eit teneste for google-brukarar som gjer det mulig å få tilgang til kraftige datamaskiner og køyre eigne program der. Dersom du syns det går for seint på di eiga datamaskin med MAX-DEPTH=4 kan du utvikle programma ferdig på di eiga maskin med MAX-DEPTH=2, og så teste programmet med MAX-DEPTH=4 på Google Colab (https://research.google.com/colaboratory).

Google Colab er nyttig òg i maskinlæringskurs, så dei som er interessert i det kan gjerne lære å bruke denne tenesta no, først som sist.

Godkjenning:

Godkjenning kan skje ved at du køyrer programmet ditt for undervisningsassistenten på laben og viser at det fungerer. Når du gjer det skal du òg vise fram tala du har fått for kor gode ComputerPlayer-subklassane er i forhold til kvarandre. Øvinga kan godkjennast på laben 18.-22. oktober. Elles må du levere ei køyrande løysing på MittUiB innan 25. okt kl 16:00.