INFO 283 – Problemløsning og søk i kunstig intelligens

Obligatorisk oppgave 21. – 31. oktober

OBLIGATORISK! Denne oppgaven må gjennomføres og godkjennes for å få gå opp til eksamen.

I denne oppgaven skal vi jobbe med et spesielt spill, og bidra til å lage et program som spiller dette spillet godt. Spillet er et såkalt null-sum-spill med to spillere som har perfekt informasjon og som spiller etter tur.

Spillet er som følgjer og kan for eksempel modellere en situasjon i et reality-show eller i en virksomhet der prosjektleder konkurrerer om status.

To spillere skal velge ut 11 team-medlemmer hver blant 22. Men hvert (prosjekt)-team har forskjellige oppgaver. Spiller 1 har fått en oppgave og spiller 2 har fått en annen oppgave. Hver kandidat har ulike evner og dermed ulik «kompatibilitet» (et tal mellom 1 og 9) til de to oppgavene. Begge spillerne har kunnskap om dette. Og ideen er at høy sum av kompatibilitet for utvalgte teammedlemmer gir høy sjanse for suksess i ditt prosjekt. Samtidig er det slik at konkurrentens suksess i hans prosjekt påverkar oppfatninga av din suksess i konteksten, så målet er å redusere konkurrentens kompatibilitet til minst mulig. *Grad av suksess i dette spillet er altså å velge ut de kandidatene som maksimerer dine resultat samtidig som det minimerer motstanderens resultat.* Differansen mellom resultatene er altså det endelige utfallet.

Dette er ikke et trivielt spill, fordi det ofte ikke lønner seg å velge den kandidaten som er best isolert sett. Man må og se på konsekvensen for motstanderen. Eksempel: En forenkla versjon med bare to kandidater har kompatibilitet som i tabellen under:

	Kandidat 1	Kandidat 2
Prosjekt 1	6	7
Prosjekt 2	9	5

I dette tilfellet vil det for prosjektleder 1 lønne seg å velge kandidat 1, slik at prosjektleder 2 ender opp med kandidat 2.

Med 22 kandidater har vi i alt 22! muligheter (1.124.000.727.777.610.000.000). Når vi konstruerer et spilltre for dette spillet med så mange mulig utfall vil det bli umulig å søke gjennom treet. Dermed kommer vi til å bruke heuristikk-funksjoner for å gjøre det mulig å søke gjennom treet på en hensiktsmessig måte.

Til å spille dette spillet har jeg laget et program med de følgende klassene:

- gjennomfører selve spillet (**SelectTeam.py**)
- en klasse som representerer informasjon om kandidater som kan velges (**Candidate.py**). Denne klassen lager og en ny instans av spillet ved å gi kandidater kompatibilitet ved hjelp av en tilfeldig tall-generator. Den er laga slik at summen av kompabilitetene for alle kandidatene er 100 for begge prosjektene/oppgavene.
- en klasse som representer informasjon om status for spillet (**GameStatus.py**), blant annet med hvem som er i trekket, hvor mange kandidater som er igjen etc.
- En abstrakt klasse Player som representere informasjon om en spiller i programmet (**Player.py**).
- En konkret klasse **HumanPlayer.py** som håndterer spillet for en menneskelig spiller.
- En abstrakt klasse **ComputerPlayer.py** som håndterer spillet for datamaskinen. Den velger trekk ved et minimax-søk gjennomført av en alphabeta-algoritme. Den implementerer ikke evalueringsfunksjonen for å vurdere hvor god en løvnode er for datamaskinen.
- **SimpleComputerPlayer.py** implementerer en enkel evalueringsfunksjon som vurderer en løvnode i spilltreet til å være summen av kompatibilitets-score'ene til de valgte kandidatene for den som er i trekket (**to_move**) minus den samme summen for den andre spilleren. Slik spillet fungerer kan det og for eksempel være verdt å se på hvor mye kompatibilitets-score som er igjen for hver av spillerene. Det er trolig en fordel å ha mer igjen enn den andre spilleren. GameStatus-klassen har en funksjon som finn kompatibilitets-scoren som er igjen for en spiller.
- **GameTreeNode.py** har det som trengs for å representere en node i speltreet slik at vi kan gjennomføre et søk. Blant annet er det denne som gjennomfører alphabeta-søket.

Oppgaven:

1. **CompVComp.py**
   Lag en versjon av **Main.py** hvor to SimpleComputerPlayer'ere spiller mot hverandre. Vi ønsker å se bort ifra bruker-input.
   Skriv kun kode i **CompVComp.py**. Ikke endre på koden i andre klasser.

   Hint: Se hvordan **select_starter()** velger hvem som starter

2. Utvid **CompVComp.py** til å spille 10 ganger. Hold styr på hvem som vinner hvert spill, og skriv ut resultatet etter alle runder er ferdig (Hvor mange ganger spiller 1 vant, hvor mange ganger spiller 2 vant og hvor mange ganger det ble uavgjort).

3. Når oppgave 2 ble kodet så starter (mest sannsynlig) spiller 1 hver gang. Dette er urettferdig! Legg inn kode slik at spiller 1 begynner alle oddetalls-spill og spiller 2 begynner alle partalls-spill.

4. Litt kjedelig å se på to helt like SimpleComputerPlayer'ere spille mot hverandre. Dermed vil vi lage en ny **SuperComputerPlayer** som er mye smartere enn **SimpleComputerPlayer**.

   Lag en ny klasse **SuperComputerPlayer.py** som er en subklasse av **ComputerPlayer**. Implementer en bedre heuristikk-funksjon enn **SimpleComputerPlayer** sin **evaluate_game_status(self, game_status)**.

   Deretter, la **SimpleComputerPlayer** spille mot **SuperComputerPlayer**. Hvor ofte vinner/taper **SuperComputerPlayer**?

   Hint: Se på hvordan **SimpleComputerPlayer.py** er implementert. Slik spillet fungerer kan det for eksempel være verdt å se på hvor mye kompatibilitets-score som er igjen for hver av spillerne. Det er trolig en fordel å ha mer igjen enn den andre spilleren. GameStatus-klassen har en funksjon som finner resterende kompatibilitets-score for en spiller.

5. Endre verdien til **MAX_DEPTH** i **GameTreeNode.py**. Hva har dette å si for kjøretiden til programmet?

Godkjenning:

Godkjenning skjer ved at du kjører programmet ditt for lab-lederen og viser at det fungerer. Når du gjør det skal du vise frem tallene på hvor ofte SuperComputerPlayer slår SimpleComputerPlayer (et krav er at SuperComputerPlayer sin heuristikk er bedre enn SimpleComputerPlayer).

Om du ikke greier å få godkjent på lab den 21. -24. oktober vil det være mulighet til å få oppgaven godkjent den 28.-31. oktober.

Siste frist for godkjenning er 16:00, 31. oktober!