TDT4113 - Datateknologi, programmeringsprosjekt

Oppgave 2: Stein, Saks, Papir

Dette dokumentet beskriver den første oppgaven i ProgLab 2, som vi kaller "Stein, Saks, Papir". For denne oppgaven gjelder at:

- Oppgaven skal løses individuelt
- Oppgaven skal løses med objektorientert kode skrevet i python
- Fristen for oppgaven er *2 uker*, dvs. implementasjonen din lastes opp på its learnming senest 20. september 2016 kl 12:00 og demonstreres senest kl 14:00 samme dag.

1 Om spillet "Stein, Saks, Papir"

I den versjonen av spillet vi skal se på er det 2 spillere, Spiller 1 og Spiller 2. Hver spiller kan velge en av tre aksjoner: "`stein`", "`saks`" eller "`papir`". I spillets første fase gjør spillerne sine valg uten å avsløre dem for hverandre. I andre fase viser de to spillerne samtidig hva de har valgt, og det bestemmes en vinner. Dersom Spiller 1 og Spiller 2 har valgt det samme er det uavgjort, hvis ikke kåres vinneren ut fra følgende regler (se også Figur 1):

- `stein` slår `saks`.
- `saks` slår `papir`.
- `papir` slår `stein`.

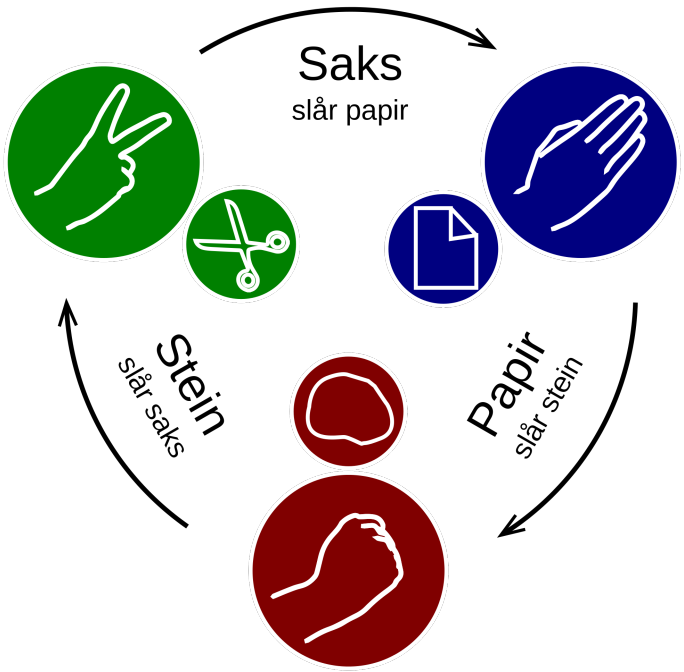

Figure 1: Hvordan bestemme vinner i Stein, Saks, Papir.

2 Implementasjon

Implementasjonen du lager skal skrives i objektorientert Python. For en rask oppfriskning av hvordan man tenker objektorientert, samt litt om hvordan vi gjør dette i Python kan du sjekke innføringsvideoen som ligger på fagets wiki-sider.

Vi skal implementere en del halv-smarte spillere til "Stein, Saks, Papir" og en test-omgivelse for å prøve dem ut. Det kan også være nyttig å utvikle noen hjelpe-klasser for å gjøre implementasjonen så enkel som mulig. Vi diskuterer disse i tur og orden nå.

2.1 Spillerne

Det er anbefalt å lage en klasse for spillere. Vi kaller den Spiller her. Spiller bør (minst) ha følgende metoder:

- velg_aksjon: Denne metoden velger hvilken aksjon som skal utføres (spille `stein`, `saks` eller `papir`) og returnerer dette.

- motta_resultat: Etter at et enkelt-spill er over får spilleren vite hva som ble valgt av begge spillere samt hvem som vant. Den kan velge å lære fra denne informasjonen (se spesielt spillerne Historiker og MestVanlig lenger ned i beskrivelsen).

- oppgi_navn: Denne metoden oppgir navnet på klassen, slik at vi kan rapportere dette i grensesnittet.

Vi skal lage et sett av spillere, og et minstekrav er å er gitt i denne listen:

- Tilfeldig: Denne velger tilfeldig om den skal gjøre `stein`, `saks`, eller `papir`. For å implementere denne kan det for eksempel være greit å bruke random.randint(0, 2). For å få tak i denne metoden må du først gjøre import random.

- Sekvensiell: Denne spilleren går sekvensielt gjennom de forskjellige aksjonene, og spiller `stein`, `saks`, `papir`, `stein`, `saks`, `papir`, `stein`, `saks` ... i en fast sekvens uansett hvordan motstander oppfører seg.

- MestVanlig: Denne spilleren ser på motstanderens valg over tid, og teller opp hvor mange ganger motstander har spilt `stein`, `saks` og `papir`. Så antar den at motstander igjen vil spille det som han/hun/den har spilt mest av så langt, og MestVanllig velger derfor optimalt ut fra dette. I det første spillet, der MestVanllig ikke har sett noen av valgene til motstander og dermed ikke vet hva den bør velge, velger den i stedet tilfeldig.

  Eksempel: La oss anta at motstander har spilt denne sekvensen: `stein`, `saks`, `stein`, `stein`, `papir`, `saks`, `papir`, `stein`, `papir`, `stein`, `stein`, `saks`, `papir`, `saks`. Vi ser at `stein` er mest vanlig i historien til motstanderen, så MestVanlig antar at `stein` vil komme igjen. Trekket fra MestVanllig er følgelig `papir` (ettersom `papir` slår `stein`).

- Historiker: Denne spilleren ser etter *mønstre* i måten motstanderen spiller på. Historiker defineres med en parameter "husk". Beskrivelsen starter med å se på situasjonen husk=1.

  Eksempel: La oss igjen anta at motstander har spilt sekvensen `stein`, `saks`, `stein`, `stein`, `papir`, `saks`, `papir`, `stein`, `papir`, `stein`, `stein`, `saks`, `papir`, `saks`. Historiker ser på det siste valget (`saks`), og går tilbake i historien for å finne hva motstanderen pleier å spille *etter* en `saks`. Motstanderen har spilt `saks` tre ganger tidligere; to av disse ble etterfulgt av `papir`, mens en ble etterfulgt av `stein`. Historiker tenker dermed at det mest vanlige etter en `saks` er `papir`, og antar derfor at den neste aksjonen til motstanderen blir `papir`. Historiker velger dermed `saks` (fordi `saks` vinner over `papir`).

Når husk er større enn 1 leter Historiker etter sub-sekvensen bestående av de husk siste aksjonene i stedet for kun å se etter den aller siste når den skal bestemme seg for hva den skal spille. Dersom husk=2 betyr det at Historiker først sjekker hvilke 2 aksjoner som ble spilt sist. I dette tilfellet var det (`papir`, `saks`), og Historiker vil derfor lete etter denne sub-sekvensen i historien. Kombinasjonen er bare spilt en gang tidligere i den rekkefølgen, og da valgte motstanderen å fortsette med `papir`. Historiker(husk=2) antar dermed at det også nå blir `papir` fra motstander etter (`papir`, `saks`), og velger følgelig selv `saks`.

Hvis vi ser på den samme sekvensen men har husk=3 må vi lete etter sub-sekvensen (`saks`, `papir`, `saks`) i historien. Denne sekvensen er ikke spilt tidligere, og Historiker skal i slike tilfeller velge tilfeldig.

2.2 Hjelpe-klasser

Det kan være nyttig å definere en Aksjon-klasse, som representerer aksjonen som velges ("Spille `stein`", "Spille `saks`", eller "Spille `papir`"). Poenget med denne klassen er å definere om en aksjon vinner over en annen. Dette gjøres ved å definere __eq__ og __gt__; se introduksjonsvideoen om objektorientert Python, spesielt om operator overloading.

Klassen EnkeltSpill kan brukes til å gjennomføre og representere et enkelt spill. EnkeltSpill brukes til å spørre etter spillernes valg, finne hvem som har vunnet, rapportere tilbake til spillerne samt rapportere til konsollet. Klassen trenger i såfall disse metodene:

- __init__(self, spiller1, spiller2): Initiere en instans av klassen, der spiller1 og spiller2 er de to spillerne.
- gjennomfoer_spill: Spør hver spiller om deres valg. Bestem resultatet ut fra regelen at det gis ett poeng til vinneren og null poeng til taperen (ved uavgjort får begge spillere et halvt poeng). Rapporter valgene og resultatene tilbake til spillerne.
- __str__: Tekstlig rapportering av enkelt-spillet: Hva ble valgt, og hvem vant?

2.3 Tekst-basert turnering for å teste to automatiske spillere

For å teste spillerne våre skal dere implementere en test-omgivelser, her definert i klassen MangeSpill. Klassen må minst ha funksjonalitet tilsvarende disse metodene:

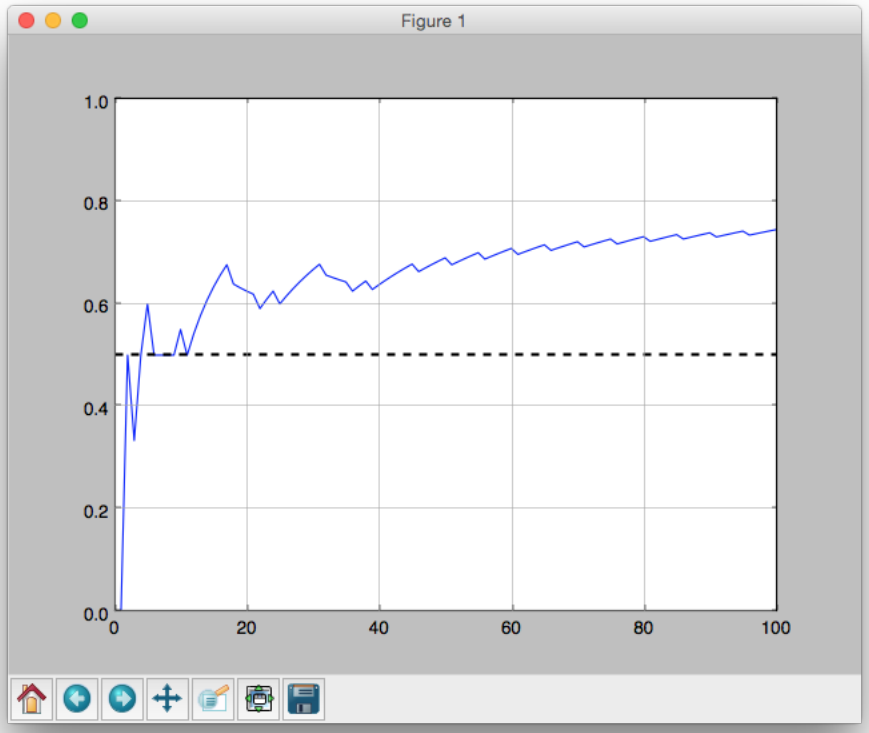

Figure 2: Utviklingen av gjennomsnittlig antall poeng per spill for Historiker(2) mot MestVanlig. Turneringen går over 100 spill og viser at Historiker(2) ganske hurtig får en god ide om hva den bør gjøre.

- __init__(self, spiller1, spiller2, antall_spill): Sette opp instansen med de to spillerne, og husker at de skal spille antall_spill ganger mot hverandre.
- arranger_enkeltspill: For å arrangere et enkelt spill må systemet spørre begge spillere om hvilken aksjon de velger, sjekke hvem som vant, rapportere valgene og resultatet tilbake til spillerne, og gi en tekstuell beskrivelse av resultatet for eksempel i form av en enkelt linje:

  ```
  Historiker(2):  Stein.  MestVanlig:  Saks -> Historiker(2) vinner
  ```

  Merk at mye av denne funksjonaliteten ligger i klassen EnkeltSpill, slik at MangeSpill.arranger_enkeltspill kan hente det meste av sin funksjonalitet derfra.
- arranger_turnering: Gjennomføre antall_spill enkelt-spill mellom de to spillerne. Rapportere gevinst-prosenten for hver av dem når turneringen er ferdig. Det

er også interessant å vise hvordan gevinst-prosenten utvikler seg over tid (se Figure 2, som viser score for Historiker(2) mot MestVanlig over 100 spill). Dette gjøres lettest ved å importere matplotlib.pyplot og bruke plot-metoden derfra.

3 Hva kreves for å bestå oppgaven

For å bestå denne oppgaven må du:

- Løse den alene innen fristen – som er på 2 uker.
- Kode de 4 Spiller-klassene Sekvensiell, Tilfeldig, MestVanlig og Historiker.
- Implementere tekst-basert grensesnitt og bruke det til å gjennomføre turneringer.

Ekstra utfordring – **ikke** obligatorisk:
Implementer din egen idé for en god Spiller. For å være "god" bør spilleren din oppnå mint 60% score mot Histroiker(2) over 1000 runder og minst mint 95% score mot Sekvensiell.