Universitetet i Bergen
Institutt for Informatikk
INF101/INF101-F
våren 2009

Obligatorisk Øving 1

Terminalbasert spill, "Studieprogresjon"

Innleveringsfrist: mandag 25.2.2008 kl. 12:00.

1 Introduksjon

Besvarelsen skal leveres elektronisk via Mi Side i studentportalen. Les kravene til innlevering nøye! Besvarelsen bedømmes av gruppelederne og foreleser i fellesskap, og det settes en poengsum på en skala fra 0 (dåligste karakter) til 100 (beste karakter). Innlevering etter fristen resulterer i poengsummen 0. En poengsum på minst 40 kreves for at oppgaven skal godkjennes. Dersom oppgaven ikke blir godkjent, kan man få lov til å levere på nytt, men poengsummen fra første innlevering blir stående. Alle de fire obligatoriske oppgavene må være bestått for å kunne ta eksamen. I så fall vil den totale poengsummen på alle de obligatoriske innleveringene utgjøre 30% av sluttkarakteren i faget.

Denne innleveringen er *individuell*. Du har lov til å diskutere løsninger med andre, men besvarelsen må formuleres selvstendig. Forøvrig viser vi til Bruk av kilder i skriftlig arbeid ved UiB og Grunnsetningar for læring på eit universitet, som du finner hos http://studiekvalitet.uib.no/ → *Etikk i utdanning*.

1.1 Forkrav

For å kunne løse disse oppgavene må du minst ha kjennskap til følgende emner:

- Objektorientert programmering (klasser, objekter, objektkommunikasjon, etc.);
- Kontrakter og abstrakte klasser;
- Tastatur I/O, med `java.util.Scanner`;
- JUnit;
- JavaDoc.

Vær nøye med utforming av kontraktene og abstrakte klasser. Systemet skal være lett å modifisere og utvides i fremtiden.

1.2 Krav til innleveringen

For at besvarelsen skal godkjennes må følgende leveres:

Oversikt Et dokument som øverst inneholder studentens fulle navn, brukernavn, og epostadresse. Videre skal dokumentet inneholde en beskrivelse av systemet, der implementasjonsvalg begrunnes. Spørsmål i oppgave 3.4 kan også besvares her. Dokumentet kan være i en av følgende åpne formater: HTML, PDF, ODT. Lukkede formater som Microsoft Word (doc) er ikke tillatt. Filnavnet skal være på formatet: `oversikt.*`.

Kildekode All kildekode skal leveres. Programkoden skal være ryddig og lett å sette seg inn i. Man kan lese mer om korrekt formatering av kode i seksjonene Introduction, Indentation, Comments and Naming Conventions fra kodekonvensjonene for Java Programming Language på `http://java.sun.com/docs/codeconv/`. Eclipse kan hjelpe til med å formatere koden din etter denne standarden. Koden skal også være dokumentert med JavaDoc.

Kjøreeksempel En fil `logg.txt` som inneholder en logg av en typisk kjøring av programmet. Et slikt kjøreeksempel viser hvordan programmet brukes og hva programmet skriver ut.

Det er svært viktig at tegnsettet til filene er lagret i enkoding UTF-8 (i Eclipse kan du sette dette ved å høyreklikke på prosjektet, velge "Properties" og justere "Text File Encoding").

Alle filene skal pakkes i en zip-fil som etter utpakking skal være mulig å kjøre også utenfor Eclipse. Spør gruppeleder om hjelp om du er usikker på hvordan du lager zip-filer.

- Mangler ved innleveringen gir trekk i poeng!
- En gjennomtestet, oversiktlig og robust implementasjon som er ukomplett er bedre enn en full implementasjon som ikke virker skikkelig. Bruk en del tid på testing av implementasjonen din.
- Hvis programmet ditt har svakheter eller mangler funksjonalitet, bør du påpeke manglene heller enn å håpe på at vi ikke oppdager dem...

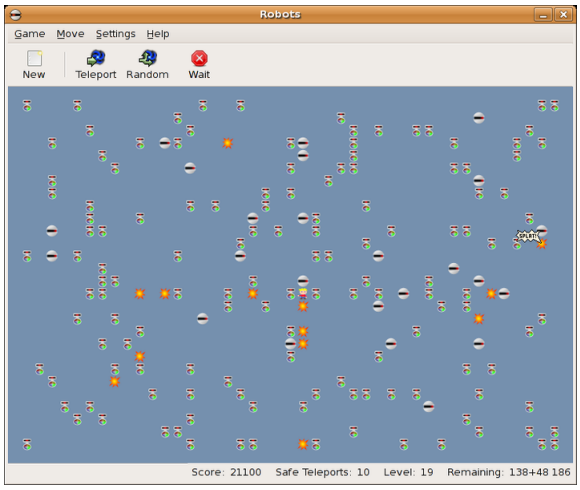
2 Terminalbasert spill

2.1 "Robots"

Spillet er basert på det klassiske spillet *Robots,* av Ken Arnold,[1] hvor man er plassert i et todimensjonalt brett (rutenett) sammen med et antall roboter. Du kan bevege deg én rute (i åtte retninger) til en ledig rute, eller evt. stå i ro hvorpå robotene blindt rykker ett steg nærmere deg. Dersom du blir innhentet av en eller flere roboter, dør du og spillet er over. Hvis to eller flere roboter kræsjer (går inn i samme rute), står det igjen et vrak i den ruten. Ytterligere roboter kan kræsje inn i dette samme vraket (hint, lurt sted å 'gjemme seg' bak). Brettet er over når du har vraket alle robotene.

Her er en versjon av Robots, som er installert på Linuxanlegget ved UiB:

[1] `http://en.wikipedia.org/wiki/Robots_(computer_game)`



2.2 "Studieprogresjon"-variant

I denne oppgaven skal du lage en variant av Robots, hvor hovedpersonen er en *student,* som jages av en mengde *forelesere.* Hvert brett er således et *semester,* hvor målet er å unngå foreleserene. Hvis to eller flere forelesere møtes, inngår de i en *diskusjon,* og er dermed inaktive. Det er alltid rom for at ytterligere forelesere blander seg inn i en pågående diskusjon...

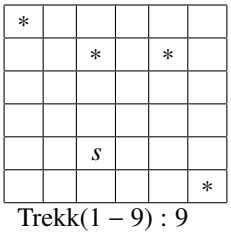
Dersom studenten blir innhentet av foreleser(e), er det tid for *veiledning,* og spillet er på den måten over. Spillet består av 6 semestre, hvor antallet forelesere øker proposjonalt med semesternummer. Studenten står fritt til når som helst å droppe ut av studiene, ved å gå utenfor brettet (semesteret). Dersom studenten overlever sitt 6. semester, har han/hun oppnådd Bachelorgrad. Gratulerer!

Hvis spillet avsluttes før Bachelorgraden er i boks, omgjøres alt stipend til studielån.

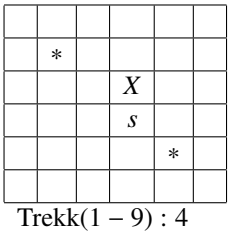
Dette spillet finnes i mange varianter (og nå, én til!), og vi kommer til å komme tilbake til dette i neste obligatoriske oppgave. I denne omgangen, utvikler vi ingen grafikk (GUI), og heller ingen våpen/triks for studenten. Det kan bli en tøff Studieprogresjon...

Eksempel

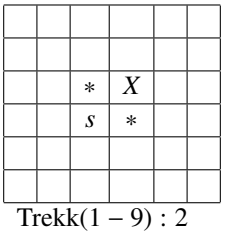
Her er et eksempel på spillforløp. Studenten vises med s, forelesere med $*$. Diskusjoner (inaktive forelesere) vises som X, mens veiledning (inaktiv student) vises med Z. Under hvert brett, ser du tasteinputet som velger studentens neste trekk (referer til det numeriske tastaturet, hvor venstre og høyre er henholdsvis 4 og 6, etc).



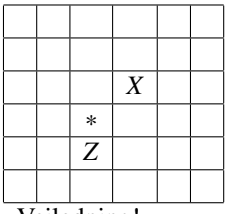
Trekk(1 – 9) : 9



Trekk(1 – 9) : 4



Trekk(1 – 9) : 2



Veiledning!

3 Oppgave

I denne oppgaven skal du implementere en terminalbasert versjon av Studieprogresjon, som skal styres med (det numeriske) tastaturet: 1 til 9 flytter deg ett steg i den aktuelle retningen (inkludert diagonalt), mens 5 lar deg stå i ro ('pass').

Oppgaven dekker alle deler av utviklingsprossen man bør gå gjennom når man skal implementere et slik brettspill. Det vil si planlegging (klassestruktur, abstrakt klasse), utføring (selve implementasjonen) og testing av koden. Oppgaven vil veilede deg gjennom hva du skal gjøre.

Studenten skal implementere Studieprogresjon-varianten av Robots, basert på følgende krav:

1. all interaksjon med systemet skal skje via konsollvinduet og tilhørende tastatur;
2. brettet skal ha variabel størrelse (`bredde` og `hoyde` settes i koden);
3. spillet skal bestå av 6 semestre (brett);
4. antall forelesere skal øke proposjonalt med semesternummer;
5. brukeren skal taste inn et nummer (1-9), etterfulgt av `Enter`-tasten, for å flytte studenten;
6. spillebrettet må vises på nytt for hvert trekk;
7. spillet skal sjekke evt. kollisjoner, både mellom forelesere, og mellom foreleser/student;
8. spillet skal avsluttes når studenten må på veiledning, eller etter at 6. semester er ferdig;

3.1 Definisjon av abstrakt klasse

Spillbrettet består av to typer *ting;* forelesere og studenter. Disse deler de fleste egenskaper, så vi lager en felles abstrakt klasse som de kan arve og utvide.

Den abstrakte klassen `Ting` skal i det minste inneholde:

```
public abstract class Ting {
   Posisjon pos;
   boolean kollidert = false;

   public Ting(Posisjon p);

   public int hentX();
   public int hentY();

   public void settPosisjon(Posisjon p);
   public Posisjon hentPosisjon();

   public boolean erKollidert();
   public void settKollidert();
   public void settIkkeKollidert();

   public boolean sjekkKollisjon(Ting[] ting);
}
```

Du må selv vurdere hvilke av disse tingene som skal være abstrakte (må implementeres i subklassene) eller som skal ha felles implementasjon (som evt. overstyres i subklassene).

Du må selv fylle inn JavaDoc-kommentarer for klassen. Metodene `hentX` og `hentY` *delegerer* til samme metoder i `Posisjon`-klassen.. Det er i hovedsak metoden `boolean sjekkKollisjon(Ting[] ting)` som krever tankearbeid. Den er beskrevet nedenfor.

3.2 Testing av Ting

For Ting-klassen kan vi tenke oss noen aksiomer som beskriver oppførselen:

- Gitt to ting `t` og `u`:

  ```
  Ting[] tabell = {u};
  if(t != u)
    assertEquals(t.hentPosisjon().equals(u.hentPosisjon()), t.sjekkKollisjon(tabell));
  ```

  dvs. `t` kolliderer med `u` hvis og bare hvis de er forskjellige objekter og har samme posisjon.

- Gitt en ting `t` og en posisjon `p`:

  ```
  t.settPosisjon(p);
  assertEquals(t.hentPosisjon(), p);
  ```

I vedlegget til oppgaven ligger det en `TingTest`-klasse med noen enkle JUnit tester for Ting klassen. Straks du har Student og Foreleser klasser, kan du bruke den til å teste implementasjonen din.

Det er lettest å kjøre JUnit-testene i Eclipse: Høyreklikk på projektet / pakken din, velg New → JUnit Test Case for å opprette en ny testklasse – kopier evt. inn koden fra TingTest. Kjør testene ved å høyreklikke på test filen og velge Run As → JUnit Test. Spør på gruppene eller forelesningen hvis du har problemer.

En oppskrift på å kjøre JUnit i DrJava finner du her: http://cnx.org/content/m11707/latest/. Å kjøre JUnit utenfor en IDE krever en god del krangling med CLASSPATH, og anbefales ikke...

3.3 Implementasjon

Følgende klasser skal implementeres:

3.3.1 Ting (abstrakt klasse) (10%)

De fleste metodene her henter eller setter verdier til og fra feltvariabler, med ett unntak:

- `boolean sjekkKollisjon(Ting[] ting)`
  Denne tar imot en tabell av ting, og den skal sjekke hvorvidt denne (`this`) har kræsjet med noe i tabellen. Som nevnt over: to ting kræsjer hvis de har samme posisjon. Hvis det skulle være flere (gruppediskusjon på gang), trenger ikke metoden finne mer enn én kollisjon, og kan returnere (`true`) så snart den finner den første.

  Metoden forholder seg til foreldreklassen `Ting`, og kan dermed brukes både for kollisjoner mellom foreleser/foreleser og foreleser/student, ved å sammenlikne posisjonsobjekter. En typisk utfordring er å unngå at metoden rapporterer at–joda–denne tingen her har kræsjet *med seg selv!* Dette skal løses.

3.3.2 Posisjon og Retning (10%)

Dette er i hovedsak en beholder for et par av koordinater, heltall `x` og `y`, men det er også en del funksjonalitet som det er praktisk å legge her:

- `void flyttMot(Posisjon p)`
  Denne metoden skal sammenlikne dette posisjonsobjektet (`this`) sine koordinater mot et annet posisjonsobjekt, `p`, og justere disse ett hakk nærmere `p`.
  Hint: `if (p.x > x) x++;`

- `void flytt(Retning retning)`
  Flytter posisjonen ett skritt i retningen *retning*.

I tillegg skal du opprette en `public equals()` metode, som kan brukes til å sammenlikne to posisjoner.

For å holde rede på retninger har vi en `enum Retning`. Du kan bruke den implementasjonen som er lagt ved oppgaven. Den definerer ni himmelretninger, `NW, N, NE, W, C, E, SW, S, SE` – `C` står for 'center,' altså ingen bevegelse. Retning har også statiske metoder for å oversette tall (fra tastaturet) til retninger, og retning til endring i x/y-posisjon.

Test av `Posisjon` Filen `PosisjonTest.java` innholder en del tester for Posisjon-klassen.

Du kan teste `flyttMot()` ved å opprette to posisjoner `p1` og `p2` med en viss avstand (tre steg, f.eks.), og kalle `p1.flyttMot(p2)` like mange ganger. Da (og bare da) skal posisjonene være like (`assertEquals(p1, p2);`).

3.3.3 Foreleser (10%)

`Foreleser` arver den abstrakte klassen `Ting`, og utvider med de nødvendige hjelpemetodene en foreleser trenger i sitt daglige virke.

- `Foreleser(Posisjon p, Ting mål)`
  Konstruktøren tar imot posisjonen til foreleseren, samt en referanse til målet (normalt sett, studenten) den skal jage.
- `String hentSymbol()`
  Skal returnere `''*''` (ett-tegns streng) for aktive foreleser, og `''X''` for inaktiv (diskuterende) foreleser.
- `void jag()`
  Oppdatér foreleserens posisjon (`pos`) vha. `flyttMot(Posisjon p)` metoden i klassen `Posisjon`. Målet til foreleseren (studenten) gis da som parameter.
  Tips: Hvordan kan du unngå at inaktive forelesere fortsetter å følge etter studenten?[2]

3.3.4 Student (10%)

`Student` arver samme klassen, `Ting`, og har også noen egne hjelpemetoder:

- `Student()` og `Student(Posisjon p)`
  Konstruktører. Vi trenger den tomme også–kommer tilbake til det (i `Spill`-klassen).
- `String hentSymbol()`
  Studenten har også to symboler; `''s''` og `''Z''`.
- `void flykt(Retning retning)`
  Kaller metoden `flytt(Retning retning)` på studentens posisjonsobjekt, `pos`, med `retning` som parameter.

3.3.5 Semester (40%)

Det meste av funksjonalitet ligger i denne klassen, som metoder.

[2] Dette skal du gjøre! Trekk for zombie-effekt i innleveringen!

Klassen har feltvariable `bredde` og `hoyde` (dimensjonene på brettet), `forelesere` (liste av forelesere), `student` (studenten), og evt. annet du finner nødvendig.

De fleste av metodene skal søke gjennom arrayet `Foreleser[] forelesere`. Følgende løkke er praktisk for dette formålet:
`for (Foreleser f : forelesere) { ... }`

Klassen `Semester` skal inneholde følgende metoder:

- `Semester(Student s, int semesterNummer, int bredde, int hoyde)`
  Konstruktøren tar imot studenten, semesternummeret, samt dimensjonene på brettet. Den tar hånd om å opprette forelesertabellen, opprette forelesere (`new`) og sette posisjonen til forelesere og student til en ledig posisjon.

- `int antallForelesere()`
  Basert på `semesterNummer`, returnerer et proposjonalt antall forelesere som du finner passende. Du må sørge for at du ikke oppretter flere forelesere enn det er plass til på brettet!
  Tips: Bruk dimensjonene på brettet, f.eks. $\lfloor$`semesterNummer` $*$ `bredde`$/5\rfloor$.

- `int antallAktive()`
  Tell opp antall aktive forelesere.

- `Posisjon finnLedigPosisjon()`
  Importér `java.util.Random`, og opprett et objekt av typen `Random`. Lag en løkke som velger tilfeldig x- og y-koordinat ($0 \leq x <$ `bredde`, $0 \leq y <$ `hoyde`) og så sjekker tabellen `forelesere` og ser hvorvidt denne posisjonen er ledig. Fortsett søket (løkken) inntil en du finner en ledig posisjon som du kan returnere.

  Hjelpemetode: `boolean ledig(Posisjon p)`. Også et tips: når du går gjennom alle foreleserene i `Foreleser[] forelesere`, sørg alltid for å sjekke at objektet faktisk *finnes* (`f != null`) før du forsøker å kalle en av dens metoder. Bruk `equals()`-metoden til å sammenlikne posisjoner.

  Tips: Skal du finne ledig posisjon til studenten, må du gjøre det etter at foreleserene er plassert – ihvertfall hvis du bare sjekker `forelesere` når du ser om en posisjon er ledig.

- `void skrivBrett()`
  Vi har ikke et brett i minne til enhver tid, så denne metoden må opprette et midlertidig brett; plassere inn alle `Ting` i rett rute; og så skrive hele greien, linje for linje, til terminalen. Husk at også inaktive forelesere skal skrives ut (det gjør ingenting om flere diskuterende forelesere plasseres i samme rute). Brettet skal også ha en ramme (bruk tegnene _ og |), og tomme ruter skal indikeres med punktum.
  Tips:

  1. Opprett et to-dimensjonalt array av typen String.
  2. Skriv ut rekke-nummer til høyre for rekkene. Det hjelper feilsøkingen din senere!

- `boolean utenforBrettet()`
  Det er mulig for studenten å flytte utenfor brettet. Denne metoden skal sjekke om dette har skjedd, og i så fall flunke vedkommende fra studiene (dvs. sette studenten inaktiv).
  Tips: Bruk `int hentX()` og `int hentY()` metodene i `Posisjon`!

- `Retning innTast(String s)`
  Skriv ut `s`, og les inn en streng fra tastaturet, og sjekk at dette er gyldig input ift. spillet; brukeren skal taste inn et *tall* mellom 1 og 9. All annen input er ugyldig, og skal føre til at brukeren får en feilmelding (f.eks. "Du må skrive et tall") og kan skrive inn en ny streng. Bruk en `switch/case` klausul til å konvertere tallverdien til en `Retning`, som returneres.
  Tips: Importér `java.io.*` eller bruk `java.util.Scanner`. Bruk en `try/catch` klausul rundt en `Integer.parseInt(String)` for å sjekke gyldighet. Du skal fange unntaket her, og skrive feilmelding.

- `boolean spill()`
  Tegner opp kartet og utfører semesterets hovedløkke:

  1. Ta imot gyldig input (tall 1-9) fra terminalen (`innTast()`), og flytt studenten med `flykt(Retning retning)`
  2. Sjekk om studenten ble tatt i veiledning (`sjekkKollisjon(forelesere)`) – sett i såfall `kollidert`-flagget hos studenten
  3. Sjekk om studenten hoppet av studiene (`utenforBrettet()`)
  4. Be hver aktive foreleser om å jage studenten ett steg
  5. Sjekk igjen om det ble veiledning
  6. Sjekk om det oppsto noen diskusjoner (for hver foreleser, `f.sjekkKollisjon(forelesere)` og evt. `f.settKollidert()`)
  7. Skriv ut brettet
  8. Hvis studenten er fortsatt aktiv og det er minst én aktiv foreleser igjen, så **gjenta** fra steg 1. Ellers er semesteret (løkken) ferdig.

  Til slutt skal metoden `true` hvis studenten fremdeles er aktiv.

3.3.6 Spill (20%)

`Spill` inneholder main-metoden og kjører selve spillet. Denne metoden skal opprette en student (med den tomme konstruktøren), og deklarere en `Semester`-variabel. Du trenger dessuten en variabel `int semesterNummer`, som initialiseres (starter) med 0.

Deretter skal følgende løkke utføres:

I Øk semesternummer med én

II Opprett nytt `Semester`-objekt, hvor konstruktøren er beskrevet overfor

III Utfør `spill()`-metoden på `Semester`-objektet

IV Hvis studenten er på veiledning , eller har droppet ut av studiene (sjekk retur fra `spill()`, er spillet over; ta pent farvel og avslutt ryddig.
Hvis `semesterNummer` er 6, har studenten fått Bachelorgrad! Det skal feires, men programmet må fortsatt avsluttes ryddig. Ellers, **gjenta** fra I med neste semester.

Hvis spillet endte innen studenten oppnådde Bachelorgrad, skal alt stipend omregnes til studielån, med følgende formel (oppdatert 2008):

$$\text{studielån} = 42500 * \text{antall semestre}$$

Dette skal skrives til skjermen, i stedet for den festlige gratulasjonsmeldingen.

3.4 Testing (bonuspoeng!)

Vi har foreslått noen tester for Ting og Posisjon. Men det kan være hensiktsmessig å teste mye annet i programmet som ledd i utviklingen:

- At `finnLedigPosisjon()` gir en ledig posisjon innenfor brettet så lenge du kaller den mindre enn *bredde* × *høyde* ganger
- At `ledig(p)` gir `false` når vi nettopp har satt en foreleser på posisjon `p`.

- At `utenforBrettet()` fungerer riktig.

1. Test programmet ditt med testene i `PosisjonTest.java` og `TingTest.java`. Hvordan gikk det? Fant du noen feil?
2. Implementer testene som er foreslått over og tidligere i oppgaven.
3. Kommer du på noe mer som hadde vært lurt å teste? Hvordan ville du gjort det?

4 Kjøreeksempel

Her er et eksempel på hvordan en kjøring av programmet kan se ut. Utskriften fra det implementerte program trenger ikke være helt lik.

```
Velkommen til Studieprogresjon!
Starter semester 1
Oppretter 4 forelesere
Poeng: 0
_____________________
|...s.............| 0
|.........*.......| 1
|.**....*.........| 2
|.................| 3
|.................| 4
|.................| 5
|.................| 6
_____________________
Flytt (1-9): 4
Diskusjon!
Poeng: 2
_____________________
|..s.....*.......| 0
|..X...*.........| 1
|.................| 2
|.................| 3
|.................| 4
|.................| 5
|.................| 6
_____________________
Flytt (1-9): 1
Poeng: 2
_____________________
|.................| 0
|.sX..*.*.........| 1
|.................| 2
|.................| 3
|.................| 4
|.................| 5
|.................| 6
_____________________
Flytt (1-9): 5
Poeng: 2
_____________________
|.................| 0
```

```
|.sX.*.*.............| 1
|....................| 2
|....................| 3
|....................| 4
|....................| 5
|....................| 6
____________________
Flytt (1-9): 5
Poeng: 2
____________________
|....................| 0
|.sX*.*.............| 1
|....................| 2
|....................| 3
|....................| 4
|....................| 5
|....................| 6
____________________
Flytt (1-9): 5
Diskusjon!
Poeng: 3
____________________
|....................| 0
|.sX.*...............| 1
|....................| 2
|....................| 3
|....................| 4
|....................| 5
|....................| 6
____________________
Flytt (1-9): 5
Poeng: 3
____________________
|....................| 0
|.sX*................| 1
|....................| 2
|....................| 3
|....................| 4
|....................| 5
|....................| 6
____________________
Flytt (1-9): 5
Diskusjon!
Poeng: 4
____________________
|....................| 0
|.sX.................| 1
|....................| 2
|....................| 3
|....................| 4
|....................| 5
|....................| 6
____________________
Alle forelesere inaktive! Ferie!!
```

```
Tast en tast for semester 2...0
Starter semester 2
Oppretter 8 forelesere
Poeng: 4
_____________________
|.....*..........s..| 0
|....................| 1
|**.........*...*....| 2
|....*.........*.....| 3
|....................| 4
|....................| 5
|.*..................| 6
_____________________
Flytt (1-9): 4
Poeng: 4
_____________________
|......*.........s...| 0
|.**.........*...*...| 1
|.....*.........*....| 2
|....................| 3
|....................| 4
|..*.................| 5
|....................| 6
_____________________
Flytt (1-9): 4
Veiledning!!
Poeng: 4
_____________________
|..**...*.....*.Z....| 0
|......*.........*....| 1
|....................| 2
|....................| 3
|...*................| 4
|....................| 5
|....................| 6
_____________________

Studielån: 85000 kroner
Game Over
```