Inlämningsuppgift 1

Version 1
”The Not So Colossal Cave”
Den första inlämningsuppgiften handlar om att kunna hantera inläst information som sedan kommer att användas i ett enkelt textbaserat spel.

1. Lite bakgrund och historia

Uppgiften är inspirerad av ett av de allra första interaktiva textbaserade spelen som gjorts. 1976 skrev Valter Crowther den första version av ”Colossal Cave” i Fortran för en PDP-10. Detta spel blev en början till en hel flora av efterföljare (Zork, Rogue, Larn, NetHack osv.). Spelet bygger på att man skall utforska ett grottsystem bestående av en massa rum där man kan finna diverse saker (guld, mat, nycklar och andra saker) och där man från varje enskilt rum kan gå i fyra olika riktningar (norr, öster, söder och väster) och ibland också upp eller ner, ibland finns det inte en dörr till ett rum i en eller flera av väderstrecken. Vinna gjorde man genom att hitta skattkistan och plocka ut dyrgriparna och ta sig tillbaka till ytan igen.
Läs mer:
https://en.wikipedia.org/wiki/Colossal_Cave_Adventure
https://en.wikipedia.org/wiki/Roguelike
Några av dessa spel är än idag under full utveckling (t ex NetHack) och trots sin enkla grafik ändå enormt roliga att spela.
Ett smakprov på hur det riktiga spelet kan se ut är: http://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure, som är en variant av Collosal Cave körbart i en webbläsare.

2. Vad programmet skall göra

2.1. Inläsning av rumsinformation

Det första steget innebär att vi från textfil, rumsinfo.txt skall läsa in information om de olika rummen och spara detta i en ArrayList. Ett rum kan beskrivas via följande egenskaper:
• rumsid, ett numeriskt värde som identifierar ett rum
• norr, innehåller ett rumsnummer som man skall komma till om man går norrut, 0 om det inte finns ett rum i denna riktning
• öster, rumsnummer för att gå åt öster, 0 om ej finns
• söder, rumsnummer för att gå åt söder, 0 om ej finns
• väster, rumsnummer för att gå åt väster, 0 om ej finns
• rumsinfo, en textsträng som beskriver rummet
• saker, i en del rum kan man hitta saker, om det inte finns något står det ”No Objects” i

annat fall vad det är för sak som finns i rummet.
En rad i textfilen ruminfo.txt kan se ut så här:Inlämningsuppgift 1
5;6;5;4;5;You are in open forest, with a deep valley to one side.;No Objects
De olika fälten enligt ovan separeras med ett semikolon (;). En rad motsvarar ett rumsobjekt.
Klassen Room har följande instansvariabler (klassen måste såklart kompletteras med ett antal instansmetoder):

```
class Room {
int roomId;
int north;
int east;
int south;
int west;
String roomInfo;
String thing;
}
```

Varje rad skall delas upp i sina beståndsdelar qoch användas för att skapa ett nytt rumsobjekt. Dessa rumsobjekt skall sedan läggas i en ArrayList<Room>. Observera att att rummen i textfilen inte är sorterade.

2.2. Vårt spel

I den första inlämningsuppgiften skall vi bara kunna gå omkring mellan rummen, fråga om det finns något objekt i rummet, och leta efter den ”The Golden Organic Banana”.
Programmet skall kunna förstå följande kommandon:

- n[orth], gå till rummet som finns i norr
- e[ast], gå til rummet som finns i öster
- s[outh], gå till rummet som finns i söder
- w[est], gå till rummet som finns i väster
- l[ook], skriv ut rumsinformationen igen
- x[examine], kolla om det finns några saker i rummet
- q[uit]. avsluta spelet
- p[ickup], plocka upp saken som finns i rummet, den upplockade saken skall inte sparas
- h[elp], skriv ut en hjälptext som beskriver alla kommandon
- Om man skriver ett felaktigt kommando så skall ett felmeddelande skrivas ut

Det skall räcka med att skriva första bokstaven i kommandot (för examine blir det den andra), men det skall också vara möjligt att skriva hela kommando-namnet.
När vi går till ett rum skall det automatiskt skrivas ut den rumsinformation som finns för detta rum. Vinna spelet gör man via att först gjort look i rummet för att se om det fanns något och sedan om saken som finns i rummet är den ”The Golden Organic Banana” så ger man kommandot pickup och ett glatt segermeddelande skrivs ut och spelet avslutas. Om man tröttnar på spelet kan man avsluta genom att ge kommandot quit.Inlämningsuppgift 1

I vilket rum som den ”The Golden Organic Banana” finns i skall slumpas fram så att mellan

varje körning av programmet så hamnar den ”The Golden Organic Banana” i ett nytt rum.
Om det redan finns något i det rum som slumpas fram skall man ersätta detta. Ni kan använda GenerateGoalState.java som finns i Inlämningsuppgift 1 mappen för att generera vilket rum som skall innehålla ”The Golden Organic Banana”.
Ni kan också använda Parser.java för kommandokoll (och TestParser2.java för ett exempel på hur man kan använda detta).

2.3. Övriga krav

Programmet med alla sina ingående java-filer skall givetvis ingå i paketstruktur precis som ni gjorde på TIG015.
I övrigt måste ni också följa den kodstandard som beskrivs i https://google.github.io/styleguide/javaguide.html.

3. Inlämning

• Uppgiften skall utföras i grupper om 2 stycken studenter, i undantagsfall 1 eller 3.
• Bokningsschema till systemtest kommer att finnas uppsatt på fönstret till Lennarts rum. Systemtest är den 20 resp. 21 februari, boka 1 pass.
• Uppgiften skall laddas upp till Gul senast 25/2 kl. 17.00. Obs. detta är efter systemtestet, till systemtestet skall ni ta med er den senaste version av ert program.
• I den zip-fil som skall laddas upp till Gul skall samtliga java-filer bifogas, en readme-fil med namn och personnummer för som utfört uppgiften.
• Om laborationsdokumentationen inkl. program ej godkänns direkt kan retur ges (max 2) på sådan dokumentation som med moderata ändringar eller kompletteringar är möjliga att godkänna. I annat fall underkänns laborationsdokumentationen utan retur.

4. Tips:

Skapa metoder för att utföra all efterfrågad funktionalitet.
Metoder som Collections.binarysearch, Collections.sort är användbara.

Inlämningsuppgift 2 - Ej applicerbar
Inlämningsuppgift 3

”The Not So Colossal Cave”
Revisited
Den tredje inlämningsuppgiften handlar om att kunna hantera inläst information till en databas som sedan kommer att användas i ett enkelt spel med ett grafisk gränssnitt.

1. Lite bakgrund och historia

Uppgiften är inspirerad av ett av de allra första interaktiva textbaserade spelen som gjorts. 1976 skrev Valter Crowther den första version av ”Colossal Cave” i Fortran för en PDP-10. Detta spel blev en början till en hel flora av efterföljare (Zork, Rogue, Larn, NetHack osv.). Spelet bygger på att man skall utforska ett grottsystem bestående av en massa rum där man kan finna diverse saker (guld, mat, nycklar och andra saker) och där man från varje enskilt rum kan gå i fyra olika riktningar (norr, öster, söder och väster) och ibland också upp eller ner, ibland finns det inte en dörr till ett rum i en eller flera av väderstrecken. Vinna gjorde man genom att hitta skattkistan och plocka ut dyrgriparna och ta sig tillbaka till ytan igen.
Läs mer:
https://en.wikipedia.org/wiki/Colossal_Cave_Adventure
https://en.wikipedia.org/wiki/Roguelike
Några av dessa spel är än idag under full utveckling (t ex NetHack) och trots sin enkla grafik ändå enormt roliga att spela.
Ett smakprov på hur det riktiga spelet kan se ut är: http://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure, som är en variant av Colossal Cave körbart i en webbläsare.

2. Vad programmet skall göra

Nu skall vi närma oss den funktionalitet som orginalspelet har, men med lite modifikationer.

2.1. Inläsning av information

Till skillnad från den första uppgiften så finns nu all information i ett antal databastabeller. Dessa tabeller finns som textfiler som ni använder för att skapa en databas. Se appendix för hur man gör om ni glömt av.
Det finns tre stycken huvudtabeller Cave, Lines och Things som motsvarar den textfil vi använde i den första uppgiften (med några utvidgningar). Dessutom finns det 2 stycken extra tabeller MinusOneRules och DragonRules (dessa används för de som vill göra lite mera, se

avsnitt 2.5).
Se appendix för exakt tabellutseende.

2.1.1. Cave

Tabellen Cave innehåller grundinformationen för ett rum:Inlämningsuppgift 3
• rumsid, ett numeriskt värde som identifierar ett rum
• norr, innehåller ett rumsnummer som man skall komma till om man går norrut
• söder, rumsnummer för att gå åt söder
• öster, rumsnummer för att gå åt öster
• väster, rumsnummer för att gå åt väster

Om något väderstreck har värdet 0 så finns det inte ett rum i denna riktning. Om det finns ett negativt värde skall ytterligare regler appliceras, se avsnitt 2.3.
Filen Cave.sql innehåller sql-satser för att skapa tabellen och sedan ett antal insert-satser för att fylla tabellen med värden. Filen innehåller också ett antal kommentarrader som beskriver vad de olika minusvärden betyder.

2.1.2. Lines

Tabellen Lines innehåller ett rumsid och rumsinformation.
Filen Lines.sql innehåller sql-satser för att skapa tabellen och sedan ett antal insert-satser för att fylla tabellen med värden.

2.1.3. Things

Tabellen Things innehåller ett rumsid och ett namn på den sak som finns i rummet (mao. så finns det inga tupler med ”No Objects” om det inte skulle finnas något i rummet). Things innehåller också ett attribut som beskriver om saken är synlig eller ej och ett attribut som beskriver om saken är åtkomlig eller ej. 1 för synlig resp. åtkomlig, 0 för osynlig resp. oåtkomlig. Mer om detta i regel-avsnittet (2.3).
Filen Things.sql innehåller sql-satser för att skapa tabellen och sedan ett antal insert-satser för att fylla tabellen med värden.

2.1.4. Gemensamt om tabellerna

Ni kan själva välja hur ni vill använda tabellerna i programmet. Finns två huvudvarianter:
1. Använda de olika tabellerna som de är i programmet, dvs. ni gör diverse SQL-satser i programmet.
2. Vid starten av programmet läsa in de olika tabellerna i lämpliga ArrayList:s, mao. liknande hur vi gjorde i inlämningsuppgift 1.
3. Eller en blandning av 1 och 2, låta någon/några tabell motsvaras av en ArrayList medans någon/några används som en databastabell.
Dessutom är det fritt fram att ändra om tabellen om ni så vill, bara ni klart dokumenterar vad

ni har gjort och varför.

2.2. Vårt spel

I den första inlämningsuppgiften skulle vi bara kunna gå omkring mellan rummen, fråga om det finns något objekt i rummet, och leta efter den ”The Golden Organic Banana”.

Bananen är nu hittats och är uppäten. Men vi skall fortfarande gå runt i de olika rummen. Nu har det också tillkommit lite nya regler om spelet. Objekt eller saker kan vara synliga eller osynliga, objekten kan också i en viss situation vara åtkomliga/användbara eller ej. Den här gången skall vi leta efter en skattkista som vi kan öppna om vi hittat alla nycklar som behövs för att låsa upp kistans fyra lås.
Precis som tidigare skall rumsinformationen skrivas ut när man går till ett nytt rum.
Programmet skall kunna förstå ett antal kommandon:
• gå till rummet som finns i norr
• gå till rummet som finns i söder
• gå till rummet som finns i öster
• gå till rummet som finns i väster
• skriv ut rumsinformationen igen
• kolla om det finns några saker i rummet
• avsluta spelet
• plocka upp saken som finns i rummet
• lägga ifrån sig ett objekt eller sak i rummet
• skriv ut en hjälptext som beskriver alla kommandon
• skriva ut innehållet i vår ryggsäck (saker vi plockat upp)
• öppna kistan (vilket bara kan göras om vi har de fyra nycklarna)
Alla kommandon skall ges via att vi klickar på en tryckknapp eller klickar på en ikon i en toolbar eller på något annat liknande sätt interagerar med det grafiska gränssnittet, vi skall mao. inte som i den första uppgiften ge textkommandon.
När vi går till ett rum skall det automatiskt skrivas ut den rumsinformation som finns för detta rum.
Mängder av ikoner till tryckknappar eller toolbar finns t ex här:
• http://www.freepik.com/free-vectors/icons
• https://www.iconfinder.com/free_icons
• http://www.flaticon.com/

2.3. Regler

Regeluppsättningen är en kraftigt förenklad variant jämfört med orginalspelet. Se kommentarerna i Cave.sql.
• Rum 13, fågeln kan bara plockas upp om vi redan plockat upp buren från rum 10. Dessutom blir fågeln rädd och flyger iväg om vi har med oss staven från rum 11.
• Rum 19, här finns en orm, som blockerar södervägen, ormen försvinner om vi har

buren med fågeln och släpper ner dem och söderrummet sätts till 29.
• Rum 34, går vi norr så dör vi och spelet avslutas.
• Rum 113, går vi åt väst så dör vi och spelet avslutas.
• Rum 120. I rummet finns en drake. Om vi släpper ner guld, juveler, diamanter och silver försvinner draken och västriktningen sätts till 0. Slutligen dyker en glasnyckel upp.
• Rum 250, om vi har alla de fyra nycklarna så kan vi öppna kistan och vi har vunnit.

2.4. Gränssnitt

Spelets användargränssnitt skall skrivas m.h.a. Java Swing, databaskopplingen med hjälp av JDBC. Hur ni designar gränssnittet är upp till er egen fantasi, En lösning kan vara att vi har ett huvudfönster bestående av 3 separata delar. En del för meddelande från spelet (rumsinfo osv.), en del som beskriver vad vi plockat upp (vår ryggsäck) och en del där vi ha spelets kommandon i form av t ex tryckknappar, ikoner i en toolbar eller något annat.

2.5. För den som vill göra mer (inte obligatoriskt)

1. Implementera en tjuv, dvs. 1 gång per xxx steg så finns det en chans att en tjuv dyker upp och som tar en sak från vår ryggsäck och sedan lägger denna i ett av rummen. vilket som helst.
2. Äta bör man annars dör man. Dvs. implementera en hungermätare som minskas med 1 för varje kommando och när denna når 0 så dör vi. Låt ryggsäcken innehålla ett bröd värt 50 enheter. För varje gång vi kommer till ett nytt rum så är chansen 1 på XX att det finns något ätbart där att plocka upp. Vi behöver också ett nytt kommando, äta.
3. Spara spelet, dvs. i vilket rum vi nu är i och vilka saker som vi plockat upp och var ifrån de plockades upp.
4. Ladda in ett sparat spel, se till att de saker som vi plockat upp från rum inte ligger kvar utan verkligen bara finns i vår ryggsäck.
5. Implementera MinusOneRules.sql och DragonRules,sql. Detta kräver att vi också implementerar ett enkelt Hitpoint system där vi startar med hälsan 50 som max och hälsan regenereras med 1 för varje steg vi tar upp till max 50hp. För de rum som har minus 1 i en riktning så slumpas ett värde fram mellan 1 och 10 fram som sedan används för att avgöra om man dör eller får en reducering i hp. Se också de speciella reglerna för drakrummet i DragonRules.sql (-3). Utöka gärna listan med rum där det kan inträffa något som reducerar din hp.

2.6. Övriga krav

Programmet med alla sina ingående java-filer skall givetvis ingå i paketstruktur precis som ni gjorde på TIG015. Ett förslag till paketstruktur:
inl3/Main
inl3/DB

inl3/GUI

Och kanske några mappar till, allt beroende på hur ni väljer att indela er kod. Jättelika metoder är förbjudna!! Sikta på uppåt 40 rader per metod eller ungefär så mycket kod som samtidigt är synlig.
I övrigt måste ni också följa den kodstandard som beskrivs i https://google.github.io/styleguide/javaguide.html.

4. Tips:

Workshop onsdag den 1 mars 10-12, kommer vi att prata om strategier, tips etc.
Använd kartorna från den första inlämningsuppgiften för navigeringshjälp.

5. Appendix

5.1. Tabellbeskrivningar

```
create table Cave (
Roomid int not null primary key,
North int,
South int,
East int,
West int
);
create table Lines (
Roomid int not null,
Linenr int not null,
Line text not null,
primary key (Roomid,Linenr),
foreign key (Roomid) references Cave (Roomid)
);
create table Things (
Thing text not null primary key,
Roomid int,
Visible int CHECK (visible = 0 or visible = 1),
Attainable int CHECK (attainable = 0 or attainable = 1),
foreign key (Roomid) references Cave(Roomid)
);
create table MinusOneRules (
Nr int not null primary key,
Effect text CHECK (Effect = 'hitloss' or Effect = 'gameover'),
```

```
Hitpoints int,
Message text
);
create table DragonRules (
Nr int not null primary key,
Effect text CHECK (Effect = 'hitloss' or Effect = 'gameover'),
Hitpoints int,
Message text
);
```

Skapa databas i Sqlite3

Starta sqlite3, t ex:

```
sqlite3 cavedatabas.db
```

cavedatabas.db blir nu namnet på den nya databasen. Ge sedan i sqlite3 följande kommandon:

```
.read Cave.sql
.read Lines.sql
.read Things.sql
```

osv.