# Instruktioner slutprojekt Avancerad Java 2021 - The Game

*Efter en lång arbetsvecka så går du som vanligt och lägger dig. Efter en orolig sömn med otäcka drömmar slår du upp ögonen och inser att du inte alls är hemma. Du befinner dig i en källare med fyra rum och du är inte ensam. Irrande mellan rummen finns det tre stycken mycket märkliga märkliga och förvirrade figurer: Jason, Freddy och Ture Sventon. Källaren saknar fönster men är möblerad med låsta möbler och slängda på golvet ligger diverse saker som du kan använda för att komma därirån.*

*The game is on!*

## Allmäna instruktioner

- **LÄS** instruktionerna noga och följ dem! Instruktionerna här är det *obligatoriska*. Spelet går att bygga ut för den som så önskar men se till att uppfylla minimikraven innan ni börjar leka! Lägg inte heller för mycket tid på spelet som spel betraktat innan du fått spelmotorn på plats!

- I git hittar ni grundstruktur - om man vill kan man använda och bygga vidare utifrån den! Det är mestadels tomma klasser men där finns ett mycket enkelt Gui.

- Funktion är viktigare än skönhet. I kursmålen står ingenting om att göra ett gui eller userXP. Gör det gärna (det ÄR kul) - men se till att inte lägga för mycket tid där! Ett supersnyggt spel som inte fungerar kommer *inte* att ge godkänt.

## Arbetssätt

**Obs! Uppgiften *är* mastig och den kommer att kräva en hel del av er! Känner ni att det inte går säg till i tid så kan vi skala bort anpassa. Däremot ska alla försöka planera och göra hela till att börja med. Ni ska också regelbundet checka in vad ni gjort i git så jag har en möjlighet att följa utvecklingen och hjälpa er!**
Regelbunden incheckning av ert arbete i git är obligatoriskt - ha för vana att alltid commita och pusha när du lämnar datorn. Känner man att mamn vågar/vill så tycker jag att ni ska testa att skapa olika grenar för olika delprojekt och sedan "merga" dem. Men det är inte en kurs i git så för min del räcker det om ni pushar till master.

*Söndra och härska!* Ett projekt i den här storleken kan ni inte göra i ett svep. Bena upp i mindre delar och implementera dessa var för sig. Gillar man JUnit så använder man det för att testa. Annars testar man manuelltmed en main och printande etc. Försök att få varje del stabil innan du går vidare.

Ett bra sätt att se vad man har behöver göra, vad man har kvar och vad man jobbar med är att göra ett **Trello** över saker och ting. Ett välgjort Trello kommer att ge en bra bild av vad du har kvar att göra - och hur du ligger till tidmässigt. En bra sak är att på kortet också skriva hur mycket tid du räknar med att det ska ta. Dels ger det en bild av hur lång tid saker tar i förhållande till vad ni först tror (hint- det tar längre tid än man tror!) - och dels ger det i alla fall en grov skiss över hur mycket arbete som återstår.

Kvarvarande kurstid är utöver några (kortare?) föreläsningar och labben, två veckor vilket *borde motsvarar 70-80 arbetstimmar* Vilket borde räcka!

## Obligatoriska klasser

Nedanstående klasser SKA finnas ( och hämtar man strukturen så finns det där!). Det är självklart tillåtet och, kanske till och med nödvändigt, att ha ännu fler klasser.

- **Main** Main ska endast starta spelet genom att skapa en instans av Game!
- **Npc** Ska vara antingen ett Interface eller en Abstrakt klass och ge ramarna för våra Npc:er. I grundstrukturen är det en abstract men det är okej att byta. Tänk bara på att byta extends mot implements i subklasserna.
- **Room** Ett Room ska ha ett unikt namn, ett Inventory och sen showMetod() som beskriver det för spelaren.
- **Player** Håller ordning på var spelaren befinner sig (alltså vilket rum som ska beskrivas), samt spelarens Inventory.
- **Update** Ska implementera Runnable och starta en tråd som Regelbundet updaterar Guit utifrån vad som händer i spelet.
- **GameObject** Ska också vara abstract eller interface hantera alla "ickelevandeöbjekt i spelet (möbler, nycklar etc). GameObject ska innehålla en boolean som avgör om objektet går att ta med sig eller är fastmonterat"i rummet.
- **Container** En subklass till **GameObject** som har ett **Inventory**. Kan vara låst eller öppet.
- **Key** En subklass till **GameObject** vars objekt används för att låsa upp Containers. Välj om keyobjektet ska hålla koll på vilken instans av container den passar till eller tvärtom!
- **Inventory** Ska innehålla en array av GameObjects. Arrayen vara anpassad efter det maxantal objekt som det kan bära (Npc bör bara kunna bära ett GameObjekt åt gången medan spelaren kan bära lite fler och rummen kan innehåla rätt många innan de blir “fulla”. Givetvis säger programmet ifrån om man försöker placera fler saker än det finns plats för. Här ska mekaniken för att plocka upp, byta bort och lägga ned objekt hanteras. *ALL HANTERING AV ARRAYERNA SKA SKE MED STREAMS - det är alltså INTE tillåtet att använda ArrayLists/LinkedLists eller andra (smartare) lösningar!*

- **Person** En person är Npc - dessa ska lagras i lista av något slag och ha ett eget liv så till vida att de ska hanteras concurrent. Npc:ernas beteende bestäms av slumptal. Det rör sig mellan rummen, plockar upp, lägger ned saker. Det ska finnas en showPerson, som visar personens namn och vad denne bär på. Vill man ha något som personen bär på så kan man antingen följa efter och vänta på att objekten läggs ned eller be om att byta mot ett objekt i det egna inventory.

- **Game** Game kommer att vara motorn"i spelet. I konstruktorn kickar vi gång GUI:s och trådar. Därefter startas spelloopen som tar in och hanterar kommandon ur ett Textfield. Updaterar spelet utifrån spelarens instruktioner. Notera att Npc kommer att röra sig oavsett vad spelaren göra. I Npc-TextArean ska man kunna se vilka Npc:er som kommer och går i rummet.

## Krav på Spelet

När spelet börjar så befinner sig spelaren i något av de fyra rummen. För enkelhets skull kallar vi dem 1-4 och lägger dem på rad. Från rum 1 kan bara nå rum 2 och från rum 2 når man rum 1 och 3 och så vidare. Spelaren skriver in vad hen vill göra i fältet för inputs och trycker sedan kommit. Vill man hellre lägga till knappar för kontroll så gör man så klart det (men snöa inte in för mycket på det!). Det rum spelaren befinner sig i visas i ett TextField (vill man kan senare lägga till en bild istället).
Befinner det sig en Npc i rummet så syns denne ett särskilt Textfield). Där kan man även se vad hen bär eventuellt bär på. Metoder för att sätta text till texfield finns guit ni kan ladda ned). Väljer spelaren att plocka upp något i rummet så flyttas objektet till spelarens inventory och lägger man ned något sker det omvända. I något av rummen placeras vi en låst dörr. För att klara spelet ska man hitta en nyckel låsa upp dörren.

## NPC:erna

Ska röra sig runt i rummen med hjälp av en eller flera trådar. AI är extremt enkel. Varje gång det är dags att agera slumpas ett tal och Npc:n genomför något av följande alternativ:

- Gå till ett anliggande rum.
- Plocka upp en objekt från det rum hen befinner sig i.
- lägga ned ett objekt som hen bär på i rummet.
- säga sin unika fras.

Vill man lägga till fler alternativ så är det så klart bara att köra (men gör klart först Bygg ut sen!). Om En NPC bär på objekt spelaren vill ha så ska man kunna byta det mot ett av spelarens objekt. Detta ska bara kunna ske om spelaren och NPC:n är i samma rum!

## Sparande av spelet

Man ska kunna spara sitt spel för att fortsätta där man var vid ett senare tillfälle. Vänta med att implementera detta tills vi gått igenom hur man gör det!

## Krav -G

För att bli godkända ska det finnas ett spelbart spel. Enstaka buggar och brister får förekomma liksom att man allt inte är hundra procent implementerat. Koden är, givetvis, väl kommenterad och följer i stort kodkonventionerna för Java.

## Krav -VG

Utöver G-kraven: Samtliga delar av instruktionerna är uppfyllda och fungerar felfritt (Enstaka, mindre allvarliga buggar kan förekomma). Det finns *.txt fil som innehåller alla speldata.. (Tänk att den ska kunna bytas för att göra ett helt annat spel)
Koden är lättläst och överskådlig och följer i allt väsentligt god sed och Javas kondkonvetioner - samt har en tillhörande JavaDokumentation. Metoder, klasser, lämpliga designpatterns och strukturen är överlag genomtänkt och motiverad.