DOKUMENTATION AV 'THE GAME'

Nedanstående dokumentation beskriv i korta ordalag vad respektive klass och dess funktioner utför.

MAIN – Startar endast en session av Game.

GAME – I princip spelets "Main". Härifrån styrs de flesta funktioner och anrop till andra klasser. Game har listor, vars innehåll är de andra klassernas värden. Game skapar, namnger och ger värden till alla rum, personer, saker och boxar och lägger in dessa i listorna. Klassen startar också igång trådpoolen som konstant ser till att gui uppdateras och att npc:erna är aktiva.

METODEN GOFORWARD – ser till att spelaren rör sig till nästa rum. Spärrar till maxvärdet 3.

METODEN GOBACK – ser till att spelaren går tillbaks. Spelaren kan inte gå lägre än 0 (första).

METODEN STARTTHREADPOOL startar en trådpool som regelbundet kör instanser av klassen update, vilket uppdatera gui med nuvarande information samt flera updateNPC som är själva AI-funktionen för NPC:erna, vilken gör att de går runt, plockar upp saker, lägger ner saker samt pratar.

METODEN ITEMACTIONS lyssnar på alla inslag av hantering av sakerna i spelet, oavsett om det är byteshandel, plocka upp, släppa eller öppna dörr eller kista.

METODERNA SAVE & LOAD gör om klassen Save, som lagrar all information gällande spelare, rum och npc i sig, till en binär sträng som lagras på datorn som en fil. Load tar sen emot filens innehåll, återskapar den till save-instansen. Tar sen dess värden och uppdaterar värdena i game genom att anropa alla dessa klassers set-funktioner.

SAVE är en klass som lagrar den allra viktigast informationen i spelet, alltså all information om spelare, npc och rum och dess innehåll. Denna används för sparfunktionerna i game.

GAMEOBJECT – grunden för alla spelets saker. Namnger grundläggande variabler för spelet samtliga objekt och är superklass för container och key.

INVENTORY innehåller en samling av GameObject och kan variera i storlek. Denna klass används på många platser i spelet. Som spelarens inventory, NPC inventory samt rummens och containers inventory.

METODEN ADDOBJECT lägger till nya värden i inventory, förutsatt att det finns en tom plats. Den använder därför METODEN FINDINDEXOF för att leta efter första null-värde i listan över objekt.

METODEN CHECKEXISTS kollar om ett gameObject finns i den här inventory.

METODEN MOVEOBJECT använder addObject och removeObject för att "flytta" objekt mellan olika inventorys. Till exempel mellan spelare och rum eller npc och spelare.

METODEN GETFIRSTOBJECT är en metod som endast används för att hämta NPC:ens första objekt.

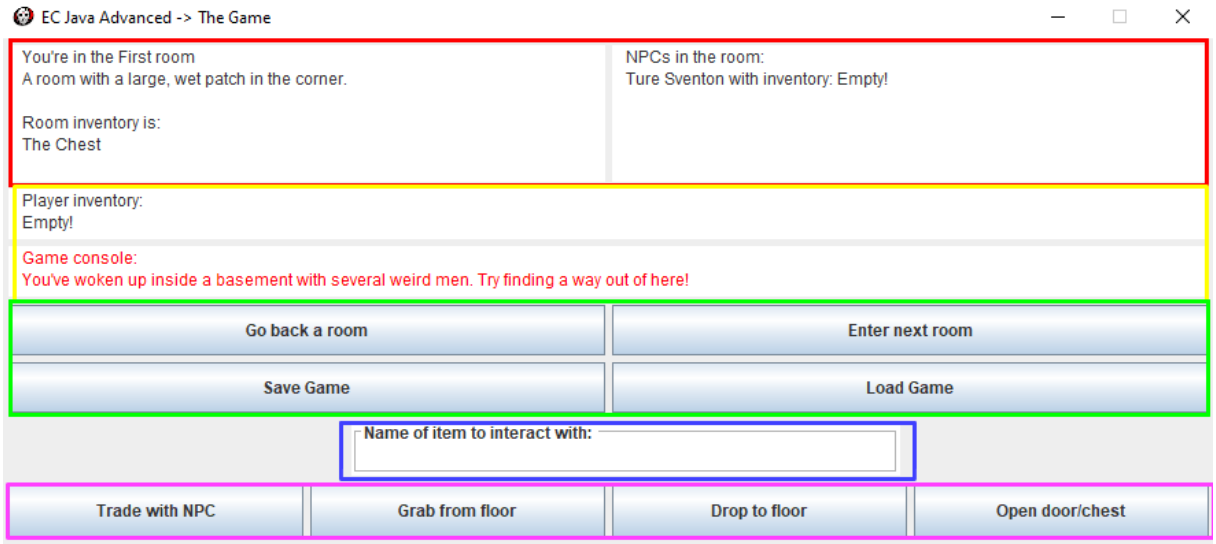
METODEN REMOVEOBJECT tar emot ett objekt, hittar första index där detta objekt finns och byter värde på den platsen till null, alltså att objektet inte finns eller syns i GUI för den delen.

METODEN TOSTRING returnerar en sträng till Gui med alla objekt i inventory-listan som inte är null. Dessa delas upp med komma-tecken. Om listan är tom visas "Empty!", vilket sätts med en ternary operator.

CONTAINER är en klass som lagrar ett inventory för lådor. Är en subklass till GameObject.

KEY är en subklass till gameObject vars funktion keyFits jämför en container med en nyckel.

GUI är den klass som visar det grafiska som syns utåt. Det finns en setShowRoom metod som tar emot en sträng och visar upp innehållet i våra rum. På samma vis finns det även setNPC som visar personerna i rummet och en setShowInventory som visar spelarens inventory samt en setConsoleLog som meddelar spelaren om vad som händer i spelet.



Jag har valt att ha flera paneler i Gridlayout-läge som kopplas samman till en helhet. (se bilden här ovan). Den röda rektangel-markeringen är en panel som beskriver rummet spelaren är i. Den visar också dess innehåll och vilka NPC:er som är där och vad de bär på.

Den gula rektangel-markeringen är vad spelaren bär på överst och undertill är information spelet skickar till spelaren. Detta kan vara vad spelet går ut på eller de slumpmässiga kommentarer NPC:er gör.

Den gröna rektangel-markeringen är knappar som inte har något alls att göra med interaktion av objekt. De har funktioner som att röra sig till nästa rum eller gå tillbaka. Nederst ser vi även spara och ladda spelet.

I den blå rektangel-markeringen skriver du namnet på det objekt du vill interagera med. Ett exempel kan vara att du vill ta upp en bit bröd, eller öppna kistan med en specifik nyckel. När du väl valt vilket objekt/sak du vill interagera med klickar du på en av de knappar som finns i den lila rektangel-markeringen. Byteshandla med NPC, plocka upp from golvet i rummet, släppa till golvet eller öppna dörr/kista.

Rent funktionsmässigt har jag en actionListener som lyssnar på de gröna knapparna, och en annan som lyssnar på de lila knapparna. De förstnämnda anropar en egen metod i Main medan de andra hämtar namn på objektet från den blå markeringen och jämför detta objekt med något annat.

NPC är en superclass för PERSON klassen och innehåller ett eget inventory.
I person-klassen lagras och sätts vilken position (rum) NPC:erna är i.

PLAYER-klassen har en liknande funktion som Person och lagrar vilket rum spelaren befinner sig i samt ett inventory.

ROOM är en klass som till stor del, rent funktionsmässigt, påminner om Inventory, då behoven är liknande blir även metoderna liknande. Det finns en addNpc, som likt addObject, frågar findIndexOf-funktionen var första tomma värdet är i lagringen och returnerar -1 och skriver ut "room is full of people" i min egna gui-console om rummet är full. Annars läggs värdet in i det första tomma indexet.

Även metoderna removeNPC och changeNPCRoom påminner om de i inventory-klassens removeObject och moveObject. I denna klass skriver toString ut rumsnamn, beskrivning och dess inventory.

UPDATE är huvuduppdateringen mellan backend (game) och gui.
Denna klass är en tråd som körs ofta. Den uppdaterar hela tiden information om rum, vilka npc som är där samt vilket inventory dessa och spelaren har.

UPDATENPC hanterar all AI / NPC-hantering och är, en av de större klasserna rent radmässigt.

Själva konstruktorn tar emot all data som kan tänkas behövas för att NPC ska fatta korrekta beslut. En ***int whichNPC*** så att man vid skapandet av tråden kan välja om man vill att detta är Jason, Freddy eller Ture (0,1,2). Se exempel nedan, där samma klass återanvänds för varje NPC, men där vissa görs mer aktiva än andra för att ge en viss personlighet.

```
Runnable thread1 = new Update(gui, basementRooms, player);
Runnable thread2 = new UpdateNpc( whichNPC: 2, items, gui, npcs, basementRooms, player);
Runnable thread3 = new UpdateNpc( whichNPC: 1, items, gui, npcs, basementRooms, player);
Runnable thread4 = new UpdateNpc( whichNPC: 0, items, gui, npcs, basementRooms, player);

tPool.scheduleAtFixedRate(thread1,  initialDelay: 0,  period: 700, TimeUnit.MILLISECONDS); //inp
tPool.scheduleAtFixedRate(thread2,  initialDelay: 1,  period: 3, TimeUnit.SECONDS); //Ture -> Qu
tPool.scheduleAtFixedRate(thread3,  initialDelay: 5,  period: 5, TimeUnit.SECONDS); //Freddy ->
tPool.scheduleAtFixedRate(thread4,  initialDelay: 10,  period: 4, TimeUnit.SECONDS); //Jason ->
```

← Denna bild är tagen från Game-klassen, men trådarna 2, 3 och 4 är UpdateNPC-klassen.

Klassen behöver också, ha kontakt till alla viktiga listor –
saker, gui, personer, rum och spelaren i sig.

Min AI fungerar lite förenklat ungefär så här.

Vid start tas följande information in:

1. Var befinner sig NPC – vilket rum?
2. Har NPC saker på sig?
3. Finns det saker på golvet?

Sen slumpas alla beslut med boolean eller int-värden.

Det finns en viss chans att NPC pratar, men bara om de är i samma rum som spelare, vilket är fristående från själva agerandet av NPC. Vad NPC säger baseras på en metod, returnQuote, som returnerar en sträng från en av NPC:ernas arrayer över kända citat de sagt. Den skrivs sen ut i gui-setConsolen. (se bild nedan)

Själva agerandet börjar med en slumpgeneratör om NPC:erna över-huvud-taget ska göra något. Om den bestämmer sig för att agera, så kollar den om det finns något på golvet i det rum den befinner sig, eller om den själv bär på något. Om det finns saker på något av dessa ställen, slumpas det om NPC ska byta rum eller ta upp/släppa saker. Bestämmer den sig sen för att gå, så slumpas det om den ska gå upp eller ner. Bestämmer den sig för saker, slumpas det om den ska droppa eller ta upp (förutsatt att båda alternativen finns).

För att förenkla, och förtydliga koden kring denna hantering, har jag skapat metoderna changeRoom, grabFromFloor och dropToFloor som gör precis vad det låter som, de byter rum på NPC, låter honom ta upp saker eller släppa saker.

```
public String returnQuote(String chosenNpcName){

    String[] tureQuotes = { "Ständigt denna Vessla", "Använd bara pitolerna i nödfall", "
    String[] jasonQuotes = { "Doomed... You're all doomed.", "Jason?", ".... [Muttering]
    String[] freddyQuotes = { "Wanna Suck Face?", "What A Rush!", "How Sweet, Fresh Meat!

    switch(chosenNpcName){
        case "Jason Voorhees" -> {
            return jasonQuotes[random.nextInt(jasonQuotes.length)];
        }
```