2018-03-02

Top-down-RPG

Kurs: Objektorienterade Applikationer (DAT 055)

Grupp 11
Emir Zivcic, zivcic@student.chalmers.se, 9607124451
Jan Rasmussen, janra@student.chalmers.se, 9602029358
Tom Bjurenlind, tombj@student.chalmers.se, 9507068493
Christer Sonesson, chrson@student.chalmers.se, 8805304659

Sammanfattning

Spelet är designat efter model-view-controller-metoden, och är uppdelat i en klient-del och en server-del. En server startas och en eller flera klienter kan då ansluta till den och spela tillsammans. Spelare kan då döda diverse monster eller attackera varandra om de så vill, målet är att få så mycket poäng som möjligt innan man till slut ger sig på slutbossen och vinner spelet. Spelarklassen användes som huvudsaklig modell och det är en av de mest innehållsrika klasserna i projektet, den håller flera viktiga objekt och innehåller många olika metoder.

Det uppnådda resultatet är ett spelbart spel med en klar början och ett klart slut, det finns en eller flera spelare som kan vinna eller förlora (genom att dö), det finns nätverksfunktionalitet, det finns möjligheter att spara och ladda spelet för att återkomma senare, det finns ljudeffekter, föremål som kan användas, utrustning som kan förbättra spelaren, olika fiender som kan attackera spelaren, samt grafik i form av .png-bilder.

Användarmanual

Spelidén är simpel: använd de tillgängliga kommandona för att navigera genom och utforska de olika rummen i spelet, döda monster, döda andra spelare, ta upp och använd föremål, och få poäng för varje dödad spelare och monster. Flera spelare kan gå med i spelet och spela samtidigt samt se och attackera varandra. Spelscenariot är att man startar i en skog fylld med monster, och ska utforska och döda dessa monster för att få poäng, samt hitta användbara föremål för detta syfte. Det finns flera nivåer och dödar man slutbossen vinner man spelet. Om man dör kastas man ut från spelet.

Då spelet är ämnat för flera spelare, måste server-programmet startas först, och sedan kan klientprogrammen köras, som då kommer ansluta till servern; förutsatt att rätt adress till servern är angiven i klientprogrammet. Kör man servern lokalt räcker det att man startar servern först, localhost är per default angiven som serveradress i klientprogrammet.

GUI layouten är även den simpel: en menyrad för att spara/ladda spelartillståndet, avsluta spelet, och för att begära en hjälptext, en bakgrundsbild för det aktuella rummet, samt bilder för monster ovanpå bakgrunden. Under bilderna finns en log där kommandon och kommandoresulat skrivs ut. Allra längst ner finns en textrad där man kan skriva in kommandon, som skickas och bearbetas av serverprogrammet.

De olika kommandona som man har till sitt förfogande är:

- Help – skriver ut en hjälptext i loggen.
- Exit – kopplar från klienten från servern och avslutar klientprogrammet.
- Back – går tillbaks till ett tidigare rum.
- Look [riktning] – skriver ut information om det aktuella rummet om ingen ”riktning” anges, så som föremål, monster, rumsbeskrivning, utgångar och spelare. Om riktning anges skrivs istället samma information ut om rummet som ligger i ”riktning” i förhållande till det aktuella rummet, om ett sådant rum existerar, t.ex. ”Look north”.
- Attack <monster> - attackerar ett angivet monster eller annan spelare, t.ex. ”Attack Orc1” eller ”Attack Player2”.
- Save <användarnamn> - sparar det nuvarande spelartillståndet under ” användarnamn”, t.ex. save ”player1”.
- Load <användarnamn> - laddar ett sparat spelartillstånd, t.ex. ”load ”player1”.
- Pick <föremål> - försöker att ta upp ett angivet föremål från det aktuella rummet, t.ex. ”Pick apple”.
- Drop <föremål> - försöker att släppa ett angivet föremål ur inventory i det aktuella rummet, t.ex. ”Drop sword”.
- Use <föremål> - försöker att använda ett angivet föremål från inventory, t.ex. ”Use pie”.
- Go <riktning> - försöker att gå ut från det aktuella rummet in i rummet mot ”riktning” i förhållande till det aktuella rummet, t.ex. ”Go north”.

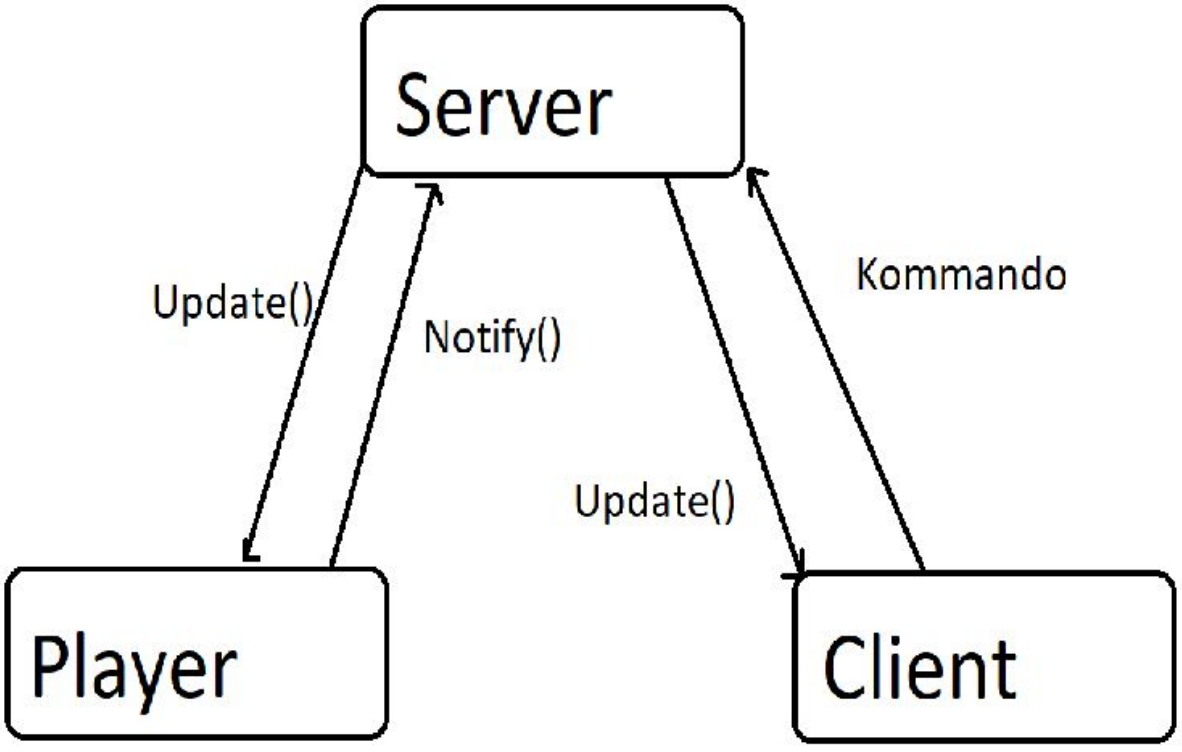
Dokumentation av modellen

Vi har använt oss utav Model-View-Controller modellen, vilket betyder att spelet vi utvecklat delats upp i tre sammankopplade delar vars primära fördel är att den interna representationen av data skiljer sig ifrån representationen av data som visas för användaren och den data som tas emot från användaren.

En utav dessa delar är paketet client, som agerar som view. I detta paket behandlas allt som angår den grafiska displayen, dvs den information som visas för användaren. Paketet har som uppgift att interagera med servern genom att förmedla data från användaren till paketet. Denna view kollar ständigt efter förändringar i modellobjektet för att presentera denna information för användaren.

Controllern i Model-View-Controller är därmed i vårat fall server-paketet. Efter att servern fått data ifrån view skickas denna data till Player-klassen som agerar som modell. Modellen kan uppdateras med t.ex. information om i vilket rum den befinner sig i, hur mycket hälsa den har kvar eller vilka föremål som finns i dennas inventory.

En vanlig interaktion mellan spelaren och spelet blir då att Vyn tar emot information från användaren, skickar den till kontrollen, kontrollen uppdaterar modellen som ständigt bevakas av vyn. Vyn uppdateras om modellen uppdaterats.



UML diagram

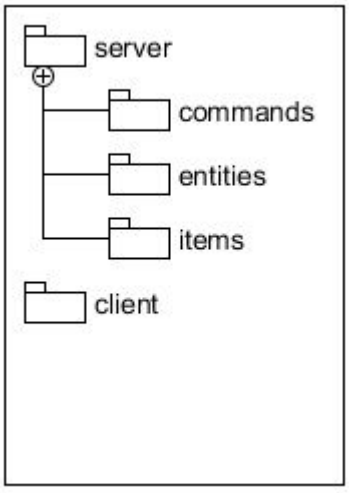


Diagram 1: Package diagram.

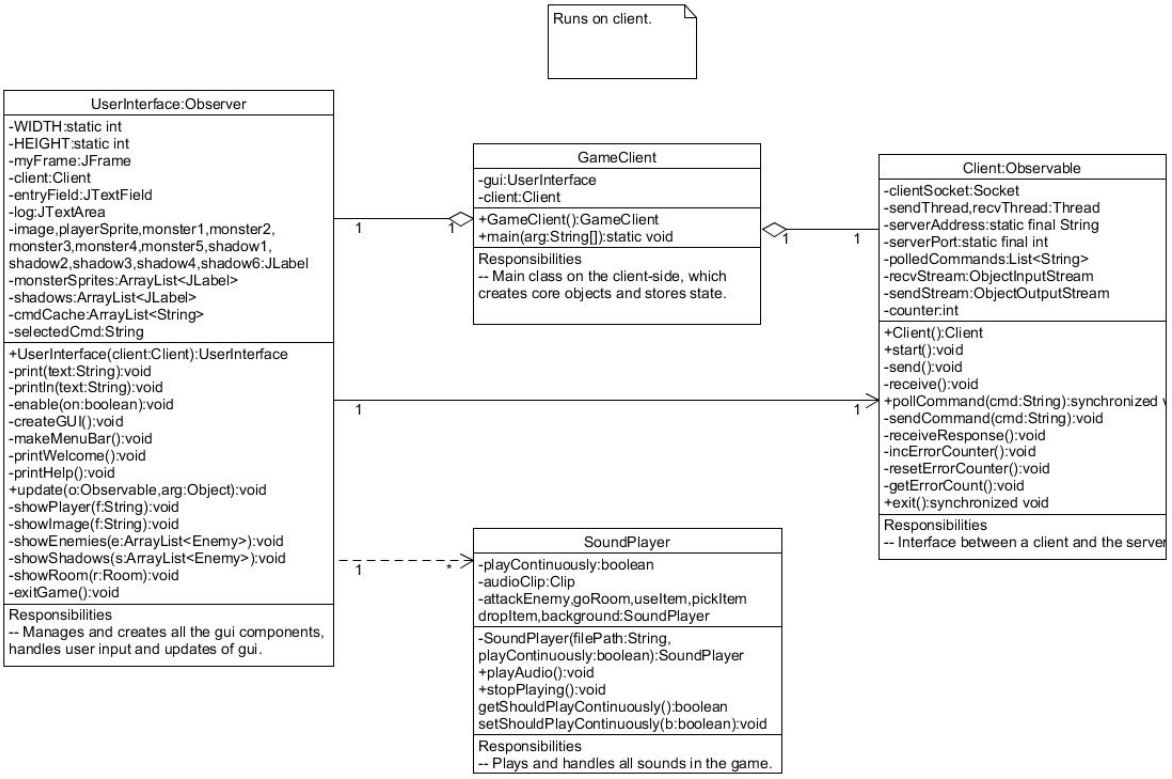


Diagram 2: Klientens klassdiagram.

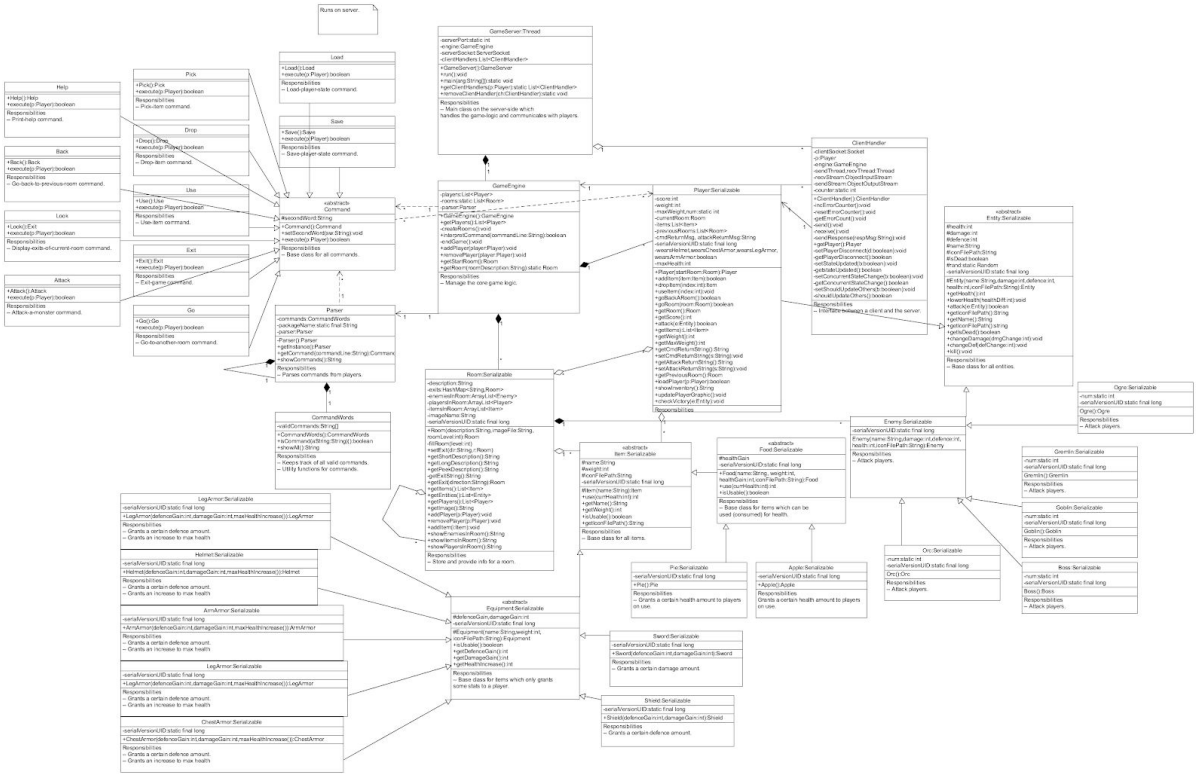


Diagram 3: Serverns klassdiagram.

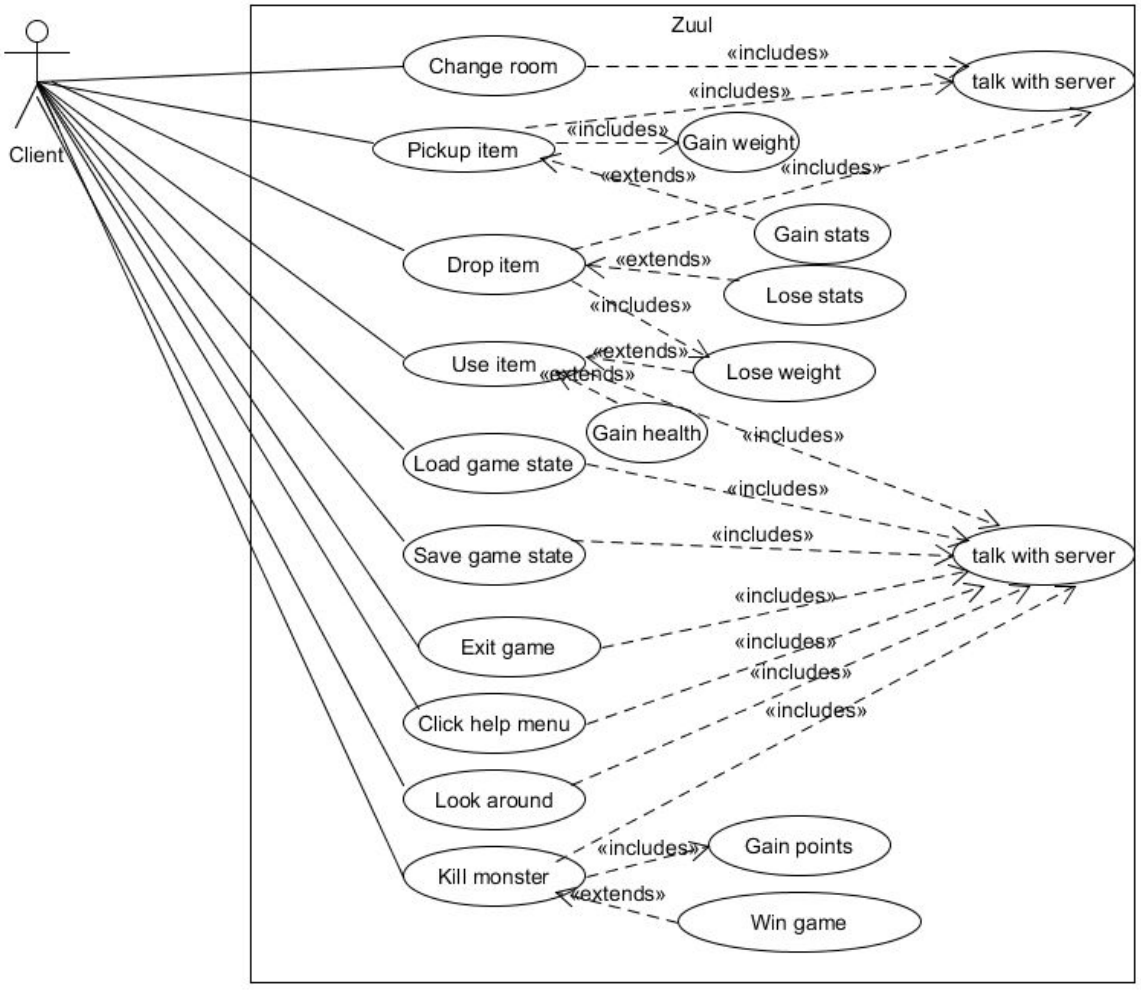


Diagram 4: use-case diagram.

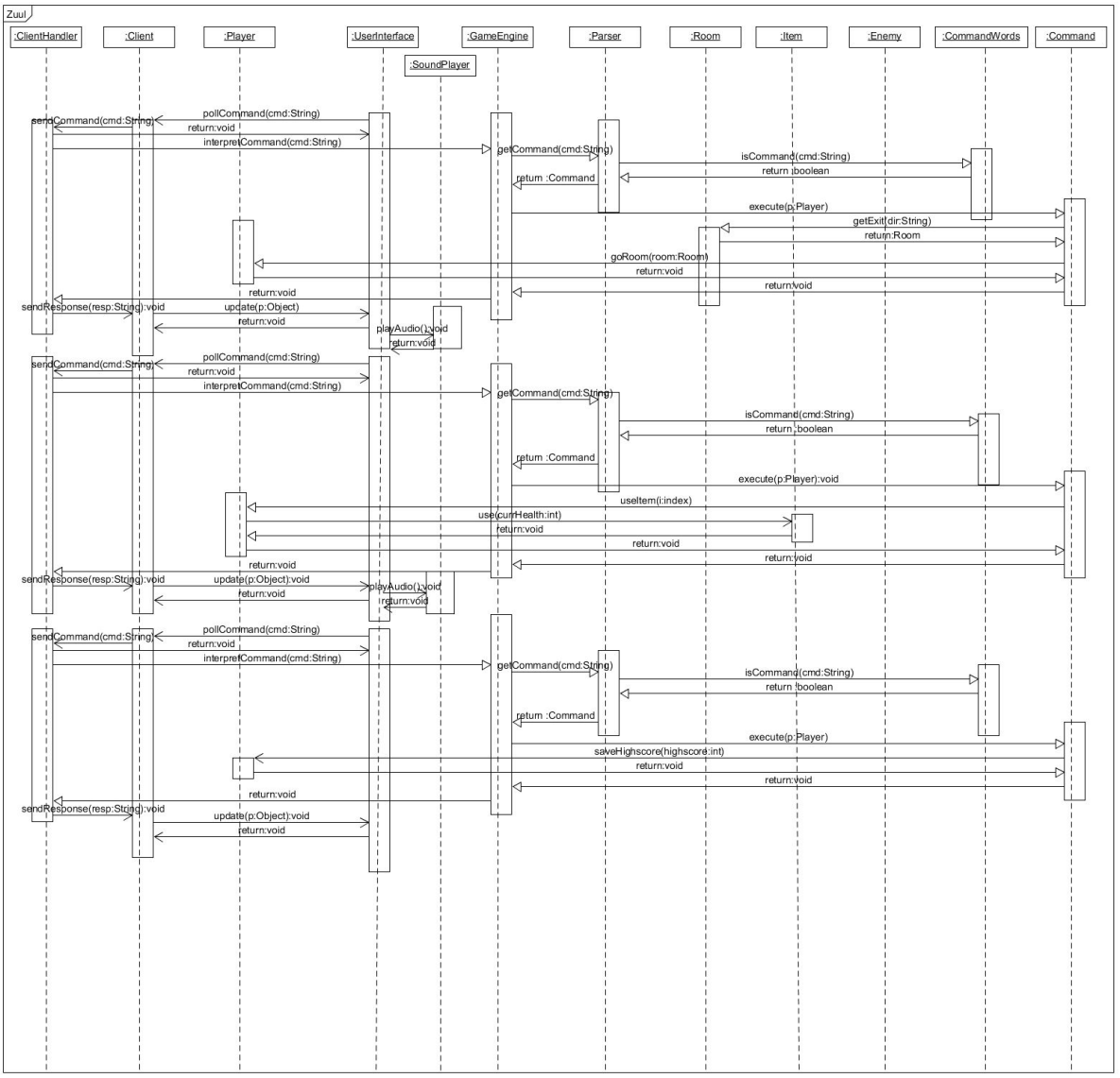


Diagram 5: Sekvensdiagram.

Klassbeskrivning

Följande klasser ligger i paketet "client", som innehåller de klasser som behövs på klientsidan av programmet.

Client:
Hanterar kommunikationen mellan klient-sidan av programmet och server-sidan.

GameClient:
Klientsidans main-klass, klassen sätter upp och startar klient-delen av programmet, den skapar de andra klasserna som behövs för att klient-programmet ska fungera.

SoundPlayer:
Har hand om allt relaterat till ljud i spelet.

UserInterface:
Har hand om allt relaterat till grafik i spelet.

Följande klasser ligger i paketet "server", som innehåller de klasser som behövs på serversidan av programmet.

ClientHandler:
Hanterar kommunikationen mellan server-sidan av programmet och klient-sidan.

GameEngine:
Skapar diverse klass-objekt som behövs för spelet, t ex alla rum, innehåller funktionalitet åt diverse andra klasser relaterat till spelar-kommandon (t ex "attack" osv)

GameServer:
Serversidans main-klass, innehåller information om vilken port som servern ska lyssna på, startar igång servern och lyssnar kontinuerligt efter nya klienter

Parser:
Klassen läser spelarkommandon (strings) och tolkar kommandot, är kommandot tillåtet så skickar den tillbaka motsvarande Command-objekt

Room:
Den här klassen representerar ett rum/en skärm i spelet, rummen har själva hand om vilka fiender och föremål som är i rummet

Följande klasser ligger i ett underpaket till "server"-paketet, som heter "server.commands". Det innehåller de klasser som representerar de kommandon spelaren kan skriva in i text-prompten

Command:
Grundklass som det är meningen att andra klasser ska 'extenda' för att få en funktionalitet, representerar ett kommando från spelaren

CommandWords:
Håller de strängar som är tillåtna för spelaren att skriva in som kommandon, har hand om att verifiera att ett skrivet kommando är tillåtet

Attack:
Klass som extendar Command, representerar kommandot "attack [enemy]", genomför en attack

Back:
Klass som extendar Command, representerar kommandot "back", gör så att spelaren backar till det föregående rummet

Drop:
Klass som extendar Command, representerar kommandot "drop [item]", gör så att spelaren släpper ett föremål

Exit:
Klass som extender Command, representerar kommandot "exit", avslutar klientsidan av spelet

Go:
Klass som extendar Command, representerar kommandot "go [direction]", flyttar spelaren till det angivna gränsande rummet

Help:
Klass som extendar Command, representerar kommandot "help", skriver upp vilka kommandon som finns tillgängliga för spelaren

Load:
Klass som extendar Command, representerar kommandot "load [filename]", laddar ett sparat spel

Look:
Klass som extendar Command, representerar kommandot "look" eller "look [direction]", skriver ut info om det rum som efterfrågas

Pick:
Klass som extendar Command, representerar kommandot "pick [item]", flyttar ett föremål från rummet till spelaren

Save:
Klass som extendar Command, representerar kommandot "save [filename]", sparar spelet

Use:
Klass som extendar Command, representerar kommandot "use [item]", genomför de effekter ett item har när det används

Följande klasser ligger också i ett underpaket till "server"-paketet, som heter "server.entities". Det innehåller de klasser som representerar fiender och spelare

Entity:
Grundklassen för spelare och fiender, innehåller de fält som behövs för att representera detta, t ex liv, attackstyrka osv

Enemy:
Abstrakt klass som extendar Entity, meningen är att specifika fiender ska extenda den här klassen, ser på så sätt till att alla fiende-klasser har en stabil 'grund' att bygga på så att det är lätt att lägga in nya fiender

Boss:
Klass som extendar Enemy, representerar ett specifikt fiende i spelet och skapar det utefter de värden som anges

Goblin:
Klass som extendar Enemy, representerar ett specifikt fiende i spelet och skapar det utefter de värden som anges

Gremlin:
Klass som extendar Enemy, representerar ett specifikt fiende i spelet och skapar det utefter de värden som anges

Ogre:
Klass som extendar Enemy, representerar ett specifikt fiende i spelet och skapar det utefter de värden som anges

Orc:
Klass som extendar Enemy, representerar ett specifikt fiende i spelet och skapar det utefter de värden som anges

Player:
Klass som extendar Entity, representerar en spelare, innehåller allt som behövs för att representera en spelare
(vilka föremål spelaren bär, vilket rum spelaren är i, osv)

Följande klasser ligger också i ett underpaket till "server"-paketet, som heter "server.items". Det innehåller de klasser som representerar föremål spelaren kan plocka upp

Item:
Representerar ett föremål spelaren kan plocka upp och använda och innehåller allt som behövs för ett generiskt föremål utan specifik funktionalitet, meningen att den ska extendas av föremål med specifika syften

Food:
Klass som extendar Item, representerar grundtypen för ett föremål som ger liv åt spelaren, innehåller det som behövs för detta.

Equipment:
Klass som extendar Item, representerar grundtypen för ett föremål som ger ökat attackvärde,
försvarsvärde, eller ökat max-liv åt spelaren och innehåller det som behövs för detta.

Apple:
Klass som extendar Food, ger lite liv åt spelaren.

Pie:
Klass som extendar Food, ger lite mer liv åt spelaren.

Shield:
Klass som extendar Equipment, ökar spelarens försvarsvärde

Sword:
Klass som extendar Equipment, ökar spelarens attackvärde

ArmArmor:
Klass som extendar Equipment, ökar spelarens försvarsvärde samt ökar max-livet

ChestArmor:
Klass som extendar Equipment, ökar spelarens försvarsvärde samt ökar max-livet

Helmet:
Klass som extendar Equipment, ökar spelarens försvarsvärde samt ökar max-livet

LegArmor:
Klass som extendar Equipment, ökar spelarens försvarsvärde samt ökar max-livet