Projektbeskrivning

Textbaserat RPG äventyrs-motor + spel

2042-03-22

Projektmedlemmar:
Arvid Rämsberg <arvra591@student.liu.se>
Oscar Ågren <oscag274@student.liu.se>

Handledare:
Bacilika Glansholm <bacgl188@student.liu.se>

Innehåll

Projektplan

1. Introduktion till projektet

Vi tänkte göra en textbaserad RPG (Role-Playing-Game) äventyrsspel-motor med medföljande spel.
Här kommer spelaren sättas i en främmande miljö beskriven av text där hen sedan kan utforska och slåss mot monster m.m.
Motorn kommer att vara utvecklad på ett sätt som gör det enkelt att skriva sina egna äventyr med egna unika items och monster.
Målet med spelet kommer vara att ta sig runt i olika platser, för att till slut döda en boss och vinna spelet

2. Ytterligare bakgrundsinformation

Varje levande varelse ska ha både HP och ATK (+ DEF, m.m om tid finns). Spelaren ska även ha en maxvikt av items den kan bära, denna vikt-gräns kan ökas genom att exempelvis ta på sig en ryggsäck eller att levla upp spelaren och sätta en status-point i rätt status.
Varje levande varelse ska ha en LVL. Vid besegrande av motståndare av x-LVL släpps y-XP som kan få spelaren att levla upp.

Combat ska ske runda för runda (Turn Based), det kommer finnas en simpel AI som styr huruvida monster attackerar eller använder exempelvis ett item.
I strid finns ett energisystem där varje action tar en viss mängd energi för att utföra, man får energi och hp varje ny runda, det går även att vila för att få tillbaka mer energi men då kan du inte attackera.

Baserat på en flagga hos fiender kan man fly från vissa strider.

Det kommer finnas items som går att konsumera (Consumables) som kan ge olika effekter, exempelvis instant health eller antidote. Dessa avlutar inte rundan om man är i strid, dessa kan även användas utanför strid.

Det kommer finnas items som går att hålla i handen (Equipables) som sedan ger en boost till spelaren/monstrens status som HP och ATK, dessa kan även ge status effekter till både den som har på sig det och den som blir attackerad.

Levande varelser kommer att gå att interagera med, detta kan vara att exempelvis attackera, inspektera eller prata med. Det kommer även finnas varelser som går att handla med.

Platser kommer kunna innehålla items, NPC:er, spelare, vägar, interagerbara objekt.

Interagerbara objekt kommer kunna ändra och lägga till nya objekt på platsen som exempelvis nya vägar och items.

Spelaren förflyttar sig genom att välja en väg av de vägar som finns på platsen.

3. Milstolpar

Spelet utvecklas utefter att motorn blir större med mer polering framåt slutet.

#	Beskrivning
1	Definera objekt som allt utgår ifrån
2	Definera varelser
3	Definera Spelare
4	Definera Items
5	Definera consumables
6	Definera Equippables
7	Definera platser
8	Bygga upp en filstruktur
9	Gör så spelaren kan inspektera objekt på en plats
10	Gör så spelaren kan förflytta sig mellan platser
11	Implementera inventory
12	Implementera stats
13	Gör så spelaren kan se sitt inventory, stats mm
14	Gör så spelaren kan prata med varelser
15	Gör så spelaren kan interagera med objekt + 15.5 så att spelaren kan

	använda objekt på fysiska saker i rummet. (ex nyckel på dörr)
16	Implementera ett combat-system
17	Consumable items funktionalitet
18	Gör så spelaren kan attackera varelser
19	Implementera ett sätt att vinna
20	Gör så spelaren kan handla
21	Implementera ett level system
22	Implementera ett skillpoint system
23	Implementera status-flaggor

4. Övriga implementationsförberedelser

Se kap 2

Ett superobjekt för allt fysiskt, dvs allt som går att inspektera, mindre superklasser som extendera superobjektet för t.ex items (Ex abstract class items extends physicsobject…)

5. Utveckling och samarbete

Vi ska skriva strukturen tillsammans (Milstolpe 1-8) Sedan delar vi upp milstolparna mellan varandra. Behöver man hjälp eller stöter på något problem medans man utvecklar så pratar man med den andra (Även om man kan lösa det själv) För att båda ska få en bättre förståelse för koden men även kunna diskutera eventuella problem med planeringen.

Projektet ska utvecklas i git och vi ska dela upp utvecklingen i olika branches. Commits ska ske ofta. Man ska INTE arbeta i main. Commit Meddelanden ska vara tydligt beskriva vad som ändras eller läggs till (ex inte “Does stuff”)

- **Har ni samma ambitionsnivå med projektet? Hur hanterar ni annars detta, om någon t.ex. främst vill komma genom kursen och få poängen, medan en**

annan vill få högsta betyg? Vill ni båda bli klara till deadline, eller vill någon arbeta vidare under nästa period eller under sommaren?

Satsar på 5

- **Vilka tider tänker ni jobba? Går ni till alla resurstillfällen eller bara vissa, och hur kommer ni i så fall överens om vilka? Vid vilka tider arbetar ni tillsammans med projektet utöver resurstillfällena? Jobbar ni ibland (tillsammans) på kvällar eller helger? Är någon av er ofrånkomligt upptagen vid vissa tider och hur löser ni detta?**

  Vi jobbar under labbtillfällen (På valfri plats) och utöver det när vi känner för det och utefter behov (Båda måste inte arbeta samtidigt eller lika lång tid, man tar den tid man behöver för att få klart arbetet) Om man vill jobba före är det viktigt att man förmedlar det och kan komma överens om arbetet. Vi ska försöka komma överens om tider man kan jobba tillsammans.

- **När ska ni ha kommit till en viss punkt i projektet? Vill ni sätta upp en egen (enkel) tidslinje för vad ni vill bli klara med och när?**

  Planen är att använda oss av Jira där vi bygger upp en plan för projektet, det kommer här bli tydligt vad som behöver vara klart när för att bli klara i tid.

- **Vad är ett godtagbart skäl för att inte kunna komma till ett ”arbetsmöte” som redan har bokats?**

  1. Ojssan jag ramlade i trappan och bröt benet, jag ligger på akuten.
  2. Kårallen va fett lit igår och jag kan knappt stå upp.
  3. Hunden åt upp cykeln
  4. Jag ska dra till portugal en vecka
  5. Jag derankade i league och måste ta igen det
  6. Sjukdom
  7. Skada
  8. Försening i lokaltrafik (ex dagens pendeltåg blev inställt)

- **Arbetar ni alltid tillsammans? Skriver en kod medan en annan tittar på? Skriver båda kod, bredvid varandra? Skriver ni vissa delar av koden ensamma? Hur ser ni till att alla gruppmedlemmar har full förståelse för all kod i projektet (vilket är ett krav)? Har någon av er speciella intressen eller kunskaper som gör er mer lämpade att arbeta med en viss del av projektet?**

Vi kommer dela upp projektet på olika sätt hänvisar till början av punkt 5

Projektrapport

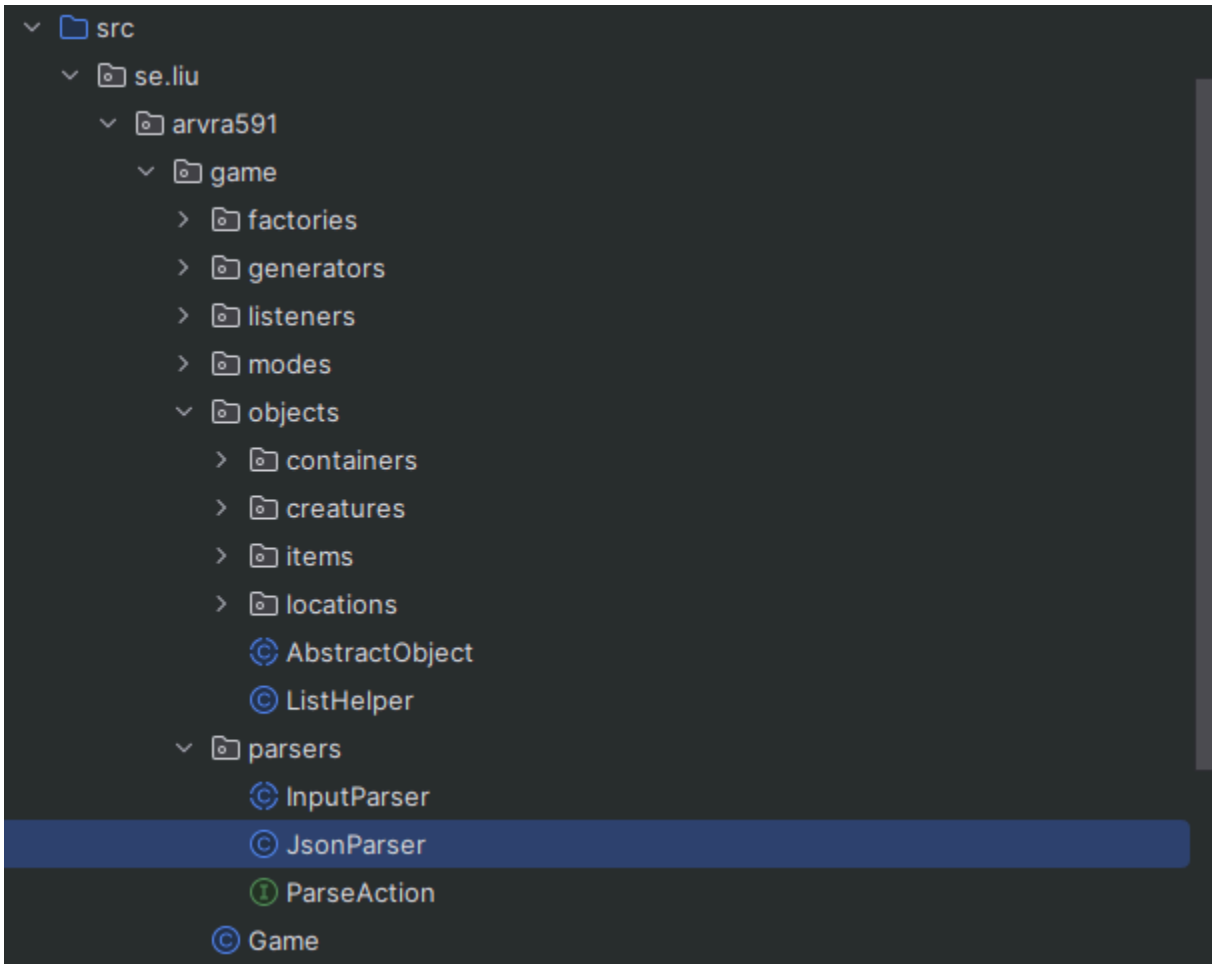
6. Implementationsbeskrivning

6.1. Milstolpar

#	Beskrivning
1	Definera objekt som allt utgår ifrån **Helt genomförd**
2	Definera varelser **Helt genomförd**
3	Definera Spelare **Helt genomförd**
4	Definera Items **Helt genomförd**
5	Definera consumables **Helt genomförd**
6	Definera Equippables **Helt genomförd**
7	Definera platser **Helt genomförd**
8	Bygga upp en filstruktur **Helt genomförd**
9	Gör så spelaren kan inspektera objekt på en plats **Helt genomförd**
10	Gör så spelaren kan förflytta sig mellan platser **Helt genomförd**
11	Implementera inventory **Helt genomförd**
12	Implementera stats **Helt genomförd**
13	Gör så spelaren kan se sitt inventory, stats mm **Helt genomförd**
14	Gör så spelaren kan prata med varelser **Helt genomförd**
15	Gör så spelaren kan interagera med objekt + 15.5 så att spelaren kan

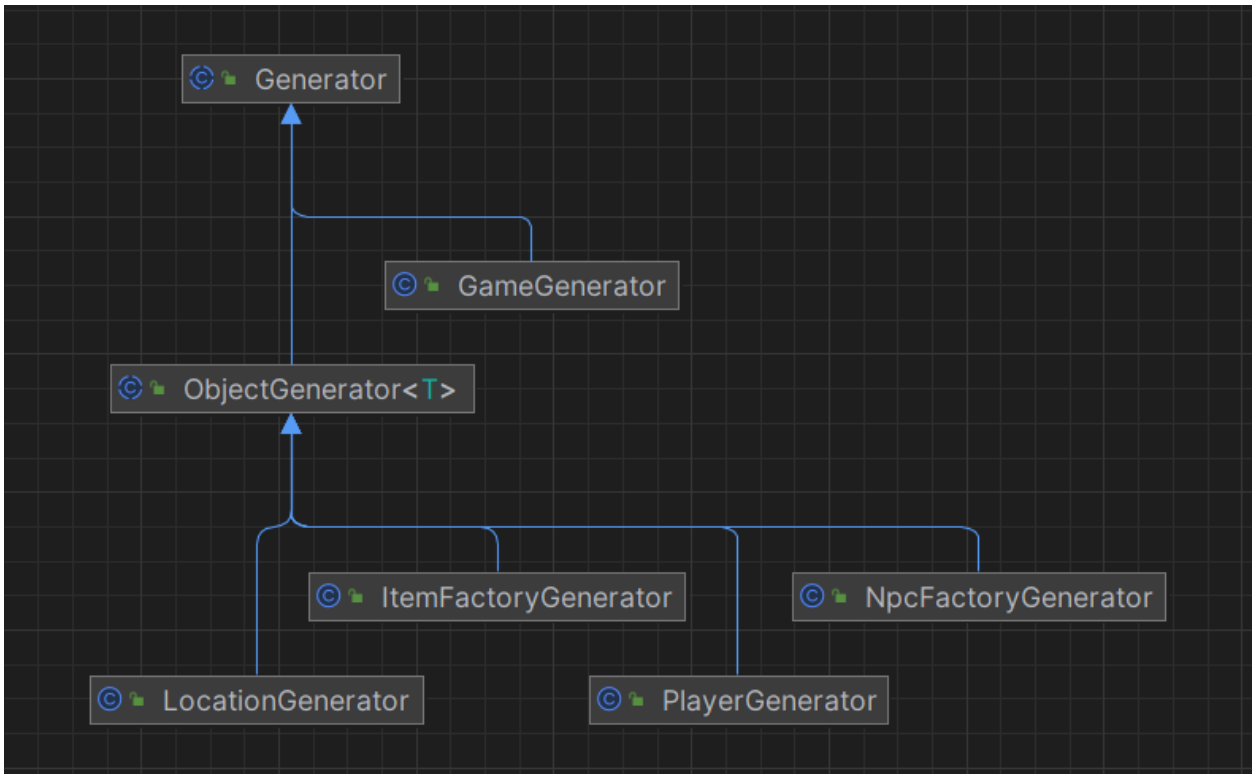
	använda objekt på fysiska saker i rummet. (ex nyckel på dörr) **Inte genomförd**
16	Implementera ett combat-system **Helt genomförd**
17	Consumable items funktionalitet **Helt genomförd**
18	Gör så spelaren kan attackera varelser **Helt genomförd**
19	Implementera ett sätt att vinna **Helt genomförd**
20	Gör så spelaren kan handla **Inte genomförd**
21	Implementera ett level system **Inte genomförd**
22	Implementera ett skillpoint system **Inte genomförd**
23	Implementera status-flaggor **Inte genomförd**

6.2. Dokumentation för programstruktur, med UML-diagram



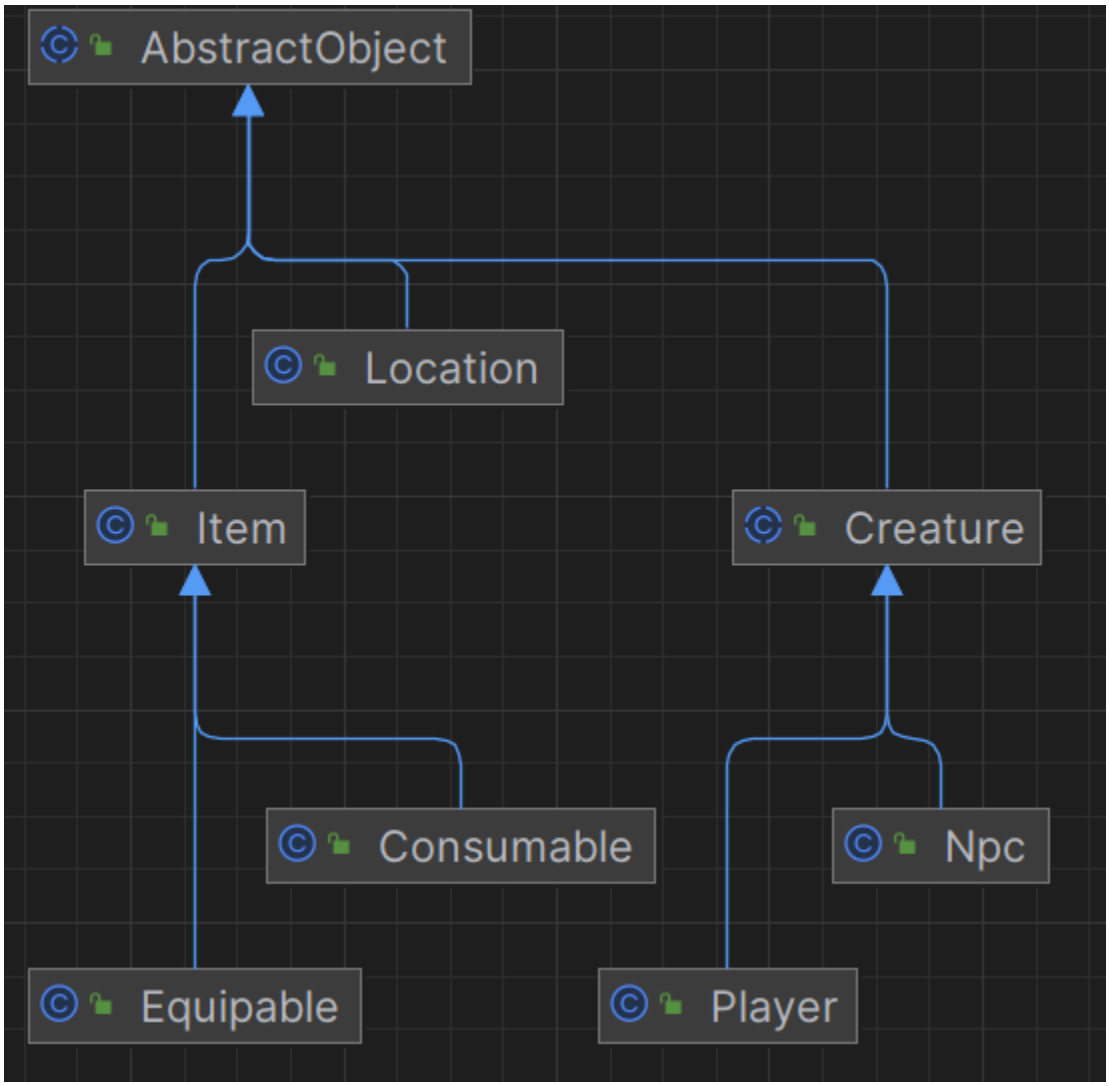
Vi har delat upp spelmotorn i flera mappar. Game är en övergripande mapp som innehåller hela spelmotorn. I game ligger factories där vi har en factory som kan generera kopior av ett huvudobjekt. I game finns också Generators som innehåller generators för olika objekt som Items, Npcs och Locations. Dessa Generators använder sig av gson för att generera objekten från givna json filer. Sedan finns game listeners som används för att kunna gå ut och in ur combat, de sköter även turordningen i combat mot NPCs och ser till att olika objekt kan tala med masterparsern. Efter det finns modes, där vi har 2 olika modes: Combat och adventure. Spelet startar i Adventure där man kan gå omkring och prata med npcs. För att attackera en npc så skriver man “engage (npc)”, då hamnar man i combat mode och här har man andra kommandon som inte finns i adventure mode såsom attack och rest. I Game klassen finns även en enum för Game States: Adventure, Combat, Win och game over som talar om för spelet vilket gamestate man är i. game har även en mapp med parsers. Det finns InputParser som används av de olika parsers som hanterar indata. Dessa implementeras i klasserna, game, Adventure och Combat. Det finns också en JsonParser som Generators använder sig av för att gå igenom json objekten för att skapa spelobjekt. UML-diagramet nedan visar klass-struckturen av alla generators, här

kan vi se hur vi har en superklass Generator som alla andra generators implementerar. GameGeneratorn har hand om att initialisera alla items, rum, varelser och spelaren i spelet. ObjectGenerator har hand om att generera objekt av typ T. Då olika objekt kräver olika implementation för att kunna genereras finns en generator för varje typ av objekt, där items och npc:er generatorerna ger factories istället för det faktiska objektet, detta för att man ska kunna ha mer än exempelvis 1 health potion.

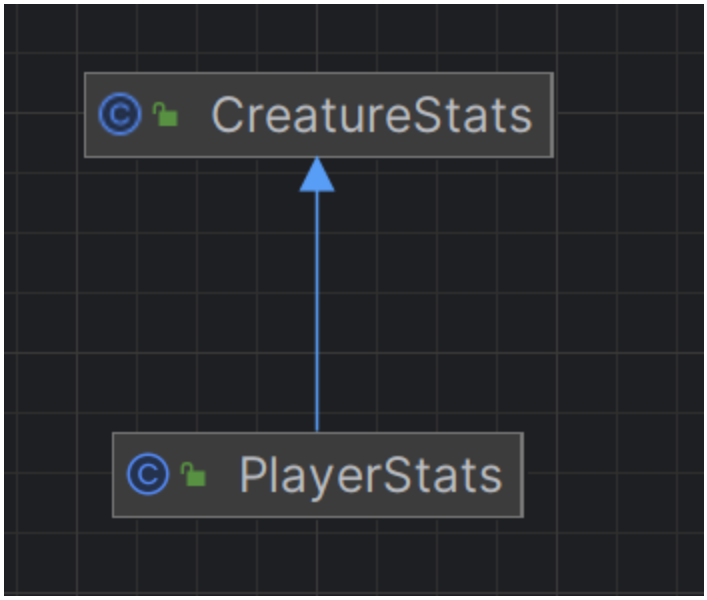


Det finns även en mapp för alla objekt i spelen. Här finns Containers som innehåller PlayerInventory och CreatureInventory som använder den generella klassen ObjectContainer. Under objects ligger även mappen creatures där vi har en abstrakt klass Creature som innehåller allt gemensamt för alla varelser i spelet. Vi har även en klass CreatureStats där de generella statsen för en creature ligger. Sedan finns det en Npc klass och en Player klass där den specifika funktionaliteten för spelaren och npcs finns. Det finns också en klass NpcsLogic som används för att npcs ska kunna attackera en spelare i combat. Sedan finns det en klass PlayerStats som är speciell för spelaren och har en carryWeight som bestämmer hur många items en spelare kan bära. Efter Creatures finns mappen Items som innehåller superklassen Item som har en vikt. Sedan finns Consumable för användbara items som har en lista med commands som skickas när man använder item. Det finns även Equipable Items som ger varelsen mer stats. Efter Items finns locations där vi har klassen Location. Location har containers av Npcs, Items och exits. Location håller även koll på om det är första gången man går in i platsen och skriver ut olika beskrivningar beroende på om det är det. I Objects ligger även en klass AbstractObject som är superklass till alla objekt. Här finns det som alla objekt ska ha såsom namn, beskrivning och möjligheten att skicka commands till spelet. Det finns även en hjälpklass ListHelper som

används för att loopa igenom listor och hitta specifika objekt genom deras namn.

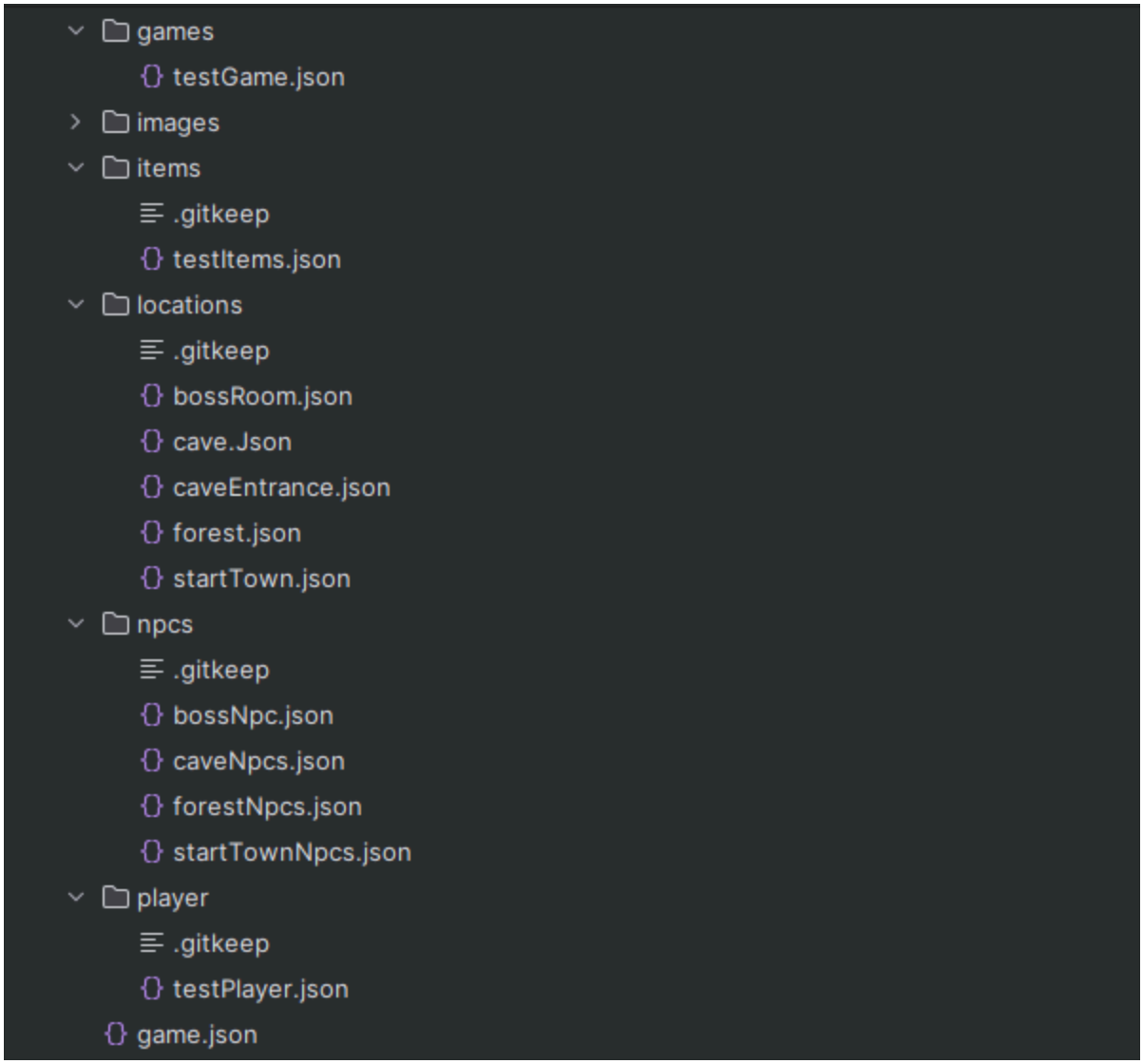


Uml diagramet ovan visar hur klass-hirarirkin är uppbyggd för objekt, vi har en superklass AbstractObject som alla objekt ärver från, vi definierar locations, creatures och items från abstractobject och sedan subklasser till dessa.



Uml diagramet ovan visar hur playerstats är en subklass till creaturestats, detta då playerstats har all data som creaturestats har plus att en spelare även har en maxvikt för vad han kan bära.

Sist i game mappen ligger klassen game. Game har en master parser som kan ta kommandon från objekt t.ex att lägga till en ny exit när en boss dör. Game tar även inputs från main klassen och skickar vidare inputs till rätt parser beroende på spelläget.

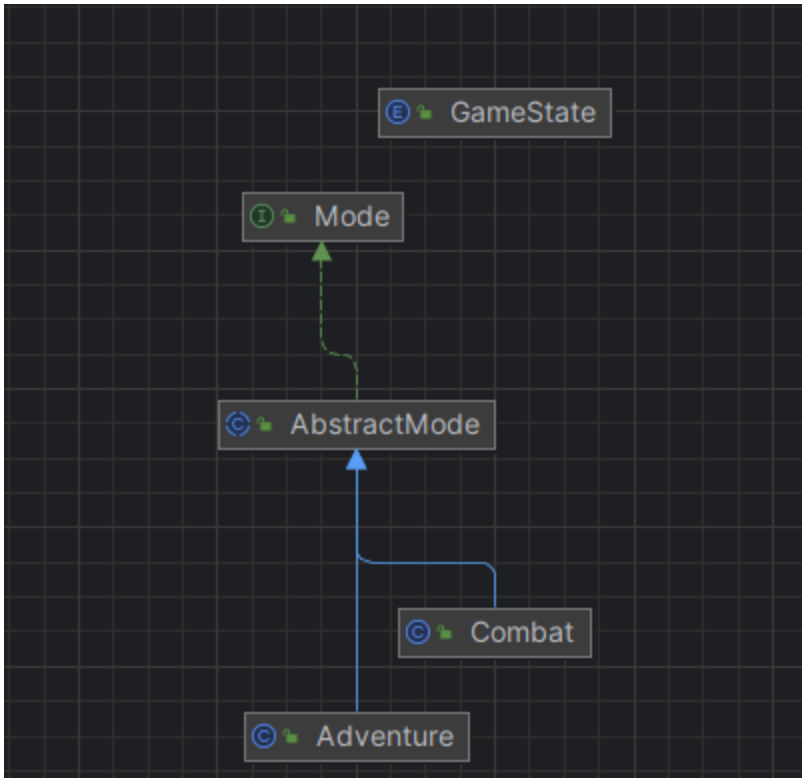


Utanför spelmotorn ligger klassen Main som initialiserar ett spel genom ett json objekt. Jsonfilerna innehåller objekt som med hjälp av generatorerna skapar spelobjekt. Detta är så att man enkelt kan skapa ett spel utan vidare förståelse för koden.

För ett komplett spel behöver man specifierat json filer som innehåller listor av alla spelare, npcs, items och platser som ska finnas i spelet.

I testItems.json finns alla items som används i testGame. Sedan finns det olika rum under locations, dessa platser har ett namn och beskrivning som alla objekt. De har även listor med npcs, items och exits. De ska även ha två listor med commands, en för första gången man går till platsen och en för de andra gångerna man kommer till platsen.

Sedan finns det en mapp med alla npcs för de olika rummen och det finns även en mapp för player.



Det här UML-Diagrammet visar hur våra game-modes fungerar. Vi har ett inteface mode som har en metod som heter parseInput. Detta interface implementeras av den abstrakta klassen AbstractMode. Combart och Adventure har egna klasser i sig som ärver av InputParser som är en abstrakt parser klass. Denna klass har en metod som också heter parseInput som tar en sträng och delar upp den i ord och kollar på första ordet, sedan jämför den ordet med en hashmap där alla kommandon finns. Om parseInput hittar en matchning så utför den en action med resten av strängen. Alltså har AbstractMode en metod parseInput som används av både Combat och Adventure klasserna. Dessa klasser har en egen parser som ärver av InputParser och kan därför använda InputParser klassens metod för att parsa strängen och hitta rätt kommando att utföra. Det finns gemensamma kommandon för båda modes som att kolla sitt inventory eller stats. Dessa ligger som protected i AbstractMode klassen och kan användas av båda klasserna. Metoderna som bara används i vardera mode är private, t.ex move och pickUp i adventure medans vissa metoder behöver vara public för att kunna användas av andra klasser. Det kan vara metoder som getCurrentTarget i Combat som kan behövas för att game ska kunna notera lyssnare att npcLogic ska använda en ny NPC som target.

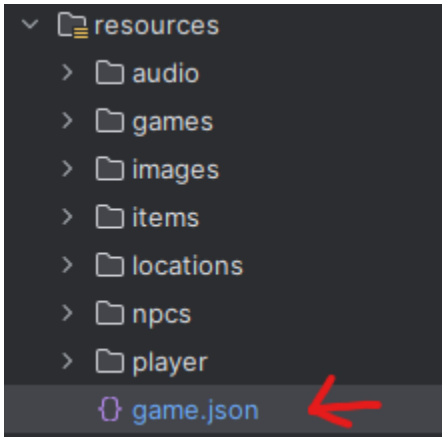
Alltså fungerar allt genom att Main tar in en input från en spelare och skickar vidare den till Game. Game använder Enumen GameState för att skicka vidare inputen till rätt Mode. I Adventure och Combats konstruktorer sätts Parser till deras respektive parser med hjälp av en setter. Sedan körs parseInput med strängen som kollar om kommandot finns och isåfall utför det.

7. Användarmanual

Motormanual

Skulle du vill skriva ditt egna textbaserade rpg-spel men känner att du inte har kunskapen att kunna skriva egen kod, då har du kommit helt rätt! Efter att ha läst igenom denna manual kommer du kunna skapa dina egna textbaserade spel som på löpande band.

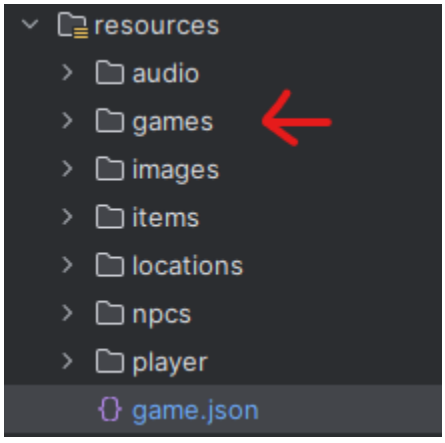
För att skapa ditt egna spel börjar du helt enkelt med att öppna json filen som heter “game.json”, denna fil ligger under “resources”.



I denna fil definierar du vilket spel du vill köra. Vill du skapa ett eget spel, ange dess namn :)

```
{
 "game": "[YOUR_GAME_NAME_HERE].json"
}
```

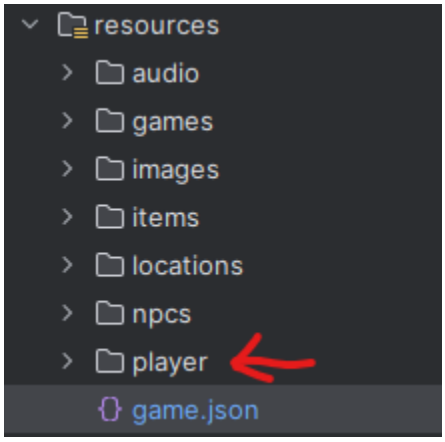
Sedan skapar du en fil under “games” med samma namn du angivit ovan, glöm inte att filens namn måste sluta på “.json” :). Du hittar mappen “games” här:



I denna fil specifierar du namnen på de filer där data för olika objekt är sparade, detta genom att följa denna struktur:

```
{
 "player": "testPlayer.json",
 "npcs": [
   "startTownNpcs.json", "forestNpcs.json", "caveNpcs.json",
"bossNpc.json"
 ],
 "items": [
   "testItems.json"
 ],
 "locations": [
   "startTown.json", "forest.json", "caveEntrance.json", "cave.json",
"bossRoom.json"
 ]
}
```

Filen för player ska ligga lagrad i mappen player som hittas enligt följande:



Denna fil ska följa följande struktur:

```
[
 {
   "name": "Kalle",
   "description": "En stark grabb",
   "health": 100,
   "energy": 100,
   "startLocation": "Start Town",
   "stats":{
     "attack": 5,
     "defense": 100,
     "maxHealth": 100,
     "maxEnergy": 100,
     "energyRegen": 0,
     "carryWeight": 10
   },
   "inventory": [
     "potion"
   ]
 }
]
```

Här definieras vad spelaren ska heta, en beskrivning av spelaren, hur mycket liv och energi han har, vart han ska börja sitt äventyr, stats, och ett inventory med items.

Sedan har vi npcs som hittas under mappen döpt npcs; dessa är definierade enligt följande struktur:

```
[
 {
 "name": "Carl",
 "description": "En vanlig grabb",
 "health": 10,
 "energy": 10,
 "stats": {
   "attack": 100,
   "defense": 1,
   "maxHealth": 10,
   "maxEnergy": 10,
   "energyRegen": 1
 },
 "dialogue": [
   [
     "say Carl: Hej!",
     "say Carl: Hoppas du mår bra :)",
     "say Carl: Här får du ett svärd",
     "giveItem sword",
     "say \ndu fick ett svärd"
   ],
   [
     "say Carl: försöker du kränga mig på fler items!?",
     "say Carl: Fy skäms!"
   ]
 ],
 "inventory": [
   "potion"
 ],
 "canDisengage": true,
 "onDeathCommands": []
}
]
```

Likt spelaren har dessa många gemensamma faktorer, men det som skiljer spelare från npcer åt är 1. Dialog, dialog definieras som en jsonarray där varje item är en egen dialog, när du pratar med npc första gången kommer första dialogens kommandon skickas till spelet för att tolkas, andra gången kommer andra dialogen att skickas osv. Mer om kommandon senare.
Det finns även en flagga som bestämmer om man kan fly från en strid med npc när man väl startat en fight, detta är användbart för att spelaren inte ska kunna fly från till exempel en påtvingad strid eller striden mot spelets sista boss. När en npc dör kommer han att droppa sina items per automatik, men du kan även specifiera några extra kommandon som sker när npc dör, exempelvis att flytta spelaren till ett annat rum eller att spelaren vinner spelet.

Sedan har vi items, items hittas under mappen “items” och definieras enligt följande struktur:

```
[
 {
   "type": "equipable",
   "name": "sword",
   "description": "Ett vanligt svärd.",
   "weight": 3,
   "stats": {
     "attack": 5,
     "defense": 0,
     "maxHealth": 0,
     "maxEnergy": 0,
     "energyRegen": 0
   }
 },
 {
   "type": "consumable",
   "name": "potion",
   "description": "En läkande dryck.",
   "weight": 1,
   "useCommands": [
     "giveplayerhealth 10",
     "say You drink the potion and gain 10 healthpoints"
   ]
 }
]
```

Här finns det två olika typer av items, equipable och cunsumable.
Ett item måste vara av en av dessa typer, och dess struktur kommer se lite olika ut baserat på vad det är för typ av item, till att börja med har alla items ett namn och en description samt en vikt.
Equipables har stats definierade, dessa stats adderas till en creatures stats när itemet är påtaget.
Consumables har en jsonarray med kommandon som sker när man använder itemet i fråga. Detta kan vara, som i exemplet ovan att ge spelaren 10hp.

Sedan har vi locations som hittas i mappen döpt “locations”, dessa är definierad efter denna struktur:

```
[
 {
   "name": "Start Town",
   "description": "Du står i en liten by mitt i en frodig skog.
Solens strålar sipprar genom trädtaken och fåglarna kvittrar glatt.",
   "npcs": [
     "Eldric",
     "Elena",
     "Carl",
     "Rufus",
     "Milo"
   ],
   "items": [],
   "exits": [
     "Forest"
   ],
   "firstEnter": [
     "say Välkommen till byn! Här i vår lilla hörna av skogen finner
du fred och vänlighet.",
     "say Du ser två NPCs som verkar vara invånare i byn. De ser
vänliga ut och ler mot dig.",
     "say Rufus, byns trogna hund, kommer skällande fram till dig och
nosar nyfiket."
   ],
   "normalEnter": [
     "say Du är återigen i den vänliga byn."
   ]
 }
]
```

Här specifierar man ett namn och beskrivning på platsen. Man specifierar alla npcs som ska finnas på platsen, alla items, och utgångar.
Vid första gången man anländer till en plats kommer kommandona under “firstEnter” köras och resterande gånger kommer kommandona under “normalEnter” att köras.

Så, nu kanske du undrar vad det finns för kommandon man kan programmera in. De kommandon som finns är följande:
“say [meddelande som ska sägas]”

“giveplayerhealth [mängd liv spelaren ska få]”
“giveplayerenergy [mängd energi spelaren ska få]”
“giveplayerattack [mängd attack spelaren ska få]”
“giveplayerdefense [mängd försvar spelaren ska få]”
“attackplayer [mängd liv spelaren ska förlora]”

“givenpchealth [mängd liv npc ska få]”
“givenpcenergy [mängd energi npc ska få]”
“givenpcattack [mängd attack npc ska få]”
“givenpcdefense [mängd försvar npc ska få]”

“moveplayer [Platts spelaren ska flyttas till]”

“giveitem [items som ska ges till spelaren]”

“disengage” tar ut spelaren ur strid
“engage [namn på npc du sätter spelaren i strid med]”
“currentmode” printar vilket läge spelare befinner sig i

“win” gör så att spelaren vinner och har därmed klarat spelet.
“lose” gör så att spelaren förlora spelet.

“spawnnpc [namn på den npc som ska spawnas i nuvarande rum”
“spawnexit [namn på den utgång som ska spawnas i nuvarande rum”
“spawnitem [namn på det item som ska spawnas i nuvarande rum”
“removenpc [namn på den npc som ska tas bort från nuvarande rum”
“removeexit [namn på den utgång som ska tas bort från nuvarande rum”
“removeitem [namn på det item som ska tas bort från nuvarande rum”

“help” printar alla möjliga kommandon.

Med dessa kommandon kommer du kunna skapa extremt invecklade och spännande världar och äventyr, bara din egen fantasi kan begränsa spelet. Lycka till!

Spelmanual

Man startar spelet genom att köra Main klassen. När spelet startas får man en beskrivning om första platsen.

```
Välkommen till byn! Här i vår lilla hörna av skogen finner du fred och vänlighet.
Du ser två NPCs som verkar vara invånare i byn. De ser vänliga ut och ler mot dig.
Rufus, byns trogna hund, kommer skällande fram till dig och nosar nyfiket.
```

För att se alla kommandon kan man skriva “help”

```
: help
Available commands:
  drop
  move
  use
  pickup
  inventory
  help
  equip
  stats
  engage
  inspect
  location
  talk
  checkinventory
```

Här ser man alla kommandon som är tillgängliga i början av spelet när man är i Adventure mode. Drop är till för att droppa Items och används genom att skriva “drop (item)”. Alla kommandon är case insensitive vilket betyder att man kan skriva med stora och små bokstäver men specifikationen av vad man vill droppa måste skrivas exakt rätt för att spelet ska kunna hålla koll på vilket objekt man menar. Move används på liknande sätt för att förflytta sig till andra platser. Use använder man med consumable items såsom en health potion, då skriver man “use potion” för att ge sin karaktär lite mer hp. Pickup gör så att man kan plocka upp items till sitt inventory och checkinventory används för att få en lista på alla items man bär på. Om man har ett equipable item i sitt inventory kan man använda equip (item) för att hålla det i sin hand och få lite extra stats. Engage använder man för att gå in i combat mode mot en specifik NPC och då skriver man engage (npc). Inspect kan man använda för att inspektera ett objekt, t.ex se en NPCs stats eller se vad som finns i en närliggande plats. Location används för att se vad som finns i platsen man är i just nu och då får man en beskrivning och listor på allt som finns där:

```
Location: Start Town
Description: Du står i en liten by mitt i en frodig skog. Solens strålar sipprar genom trädtaken och fåglarna kvittrar glatt.
Items in location:

Creatures in location:
  Eldric
  Elena
  Carl
  Rufus
  Milo

Exits in location:
  Forest
```

Man kan även prata med NPCs med hjälp av Talk kommandot.

Om man går in i combat med en NPC får man först en lista med sina egna och NPCns stats:

```
: engage Elena
You are in combat with Elena
Elena stats are:
Attack: 5
Defense: 3
Max Health: 15
Max Energy: 15
Energy Regen: 1
Current health: 15
Current energy: 15

Your stats are:
Attack: 5
Defense: 100
Max Health: 100
Max Energy: 100
Energy Regen: 0
Carry Weight: 10
:
```

I combat mode har man även några nya kommandon som rest, disengage, attack och enemyinfo.

```
: help
Available commands:
  help
  rest
  disengage
  equip
  stats
  attack
  use
  enemyinfo
  inventory
  checkinventory
```

Rest kan man använda för att vila en runda och få mer energy. Disengage kan man använda om man vill sluta slåss mot en fiende, detta kommando kan man däremot inte använda på alla NPCer så var försiktig! Attack använder man för att attackera sin fiende vilket avslutar sin tur och man får sedan veta hur rundan gick.

```
: attack
You attacked Elena for 2 damage

Elena attacked you for 0 damage

You have 100 health and 90 energy
Elena now has 11 health and 6 energy
```

Man kan också skriva enemyinfo för att få reda på vad fienden har för stats.

```
Name: Elena
Type: Npc
Description: En ung kvinna som är känd för sina berättelser om skogens magi.
Health: 11
Attack: 5
Defense: 3
Max Health: 15
Max Energy: 15
Energy Regen: 1
.
```