Projektrapport: Enkelt RPG-spel i Java
av Kerim Kozo

Beskrivning av spelet
Mitt spel är ett textbaserat RPG (role-playing game) där spelaren får välja mellan två hjälte klasser och ska ta sig igenom en dungeon med ett valfritt antal nivåer. Spelet är utformat att var femte nivå väntar en boss och man vinner när man har besegrat alla nivåer. I spelet finns även en shop att köpa vapen och liv samt finns där möjlighet att aktivera passiva abilities som hjälper en på vägen.

Planering
Varje programmerings dag började med att skissa upp ideer(brainstorm) som classer, funktioner, features e.t.c i notepad. Dessa ideer påbörjades sedan att koda och modifierades under tiden. Allt som kodades lades upp i en "projektpärm" kallad grupp på github. Det skapades även klassdiagram via mermaid för att på ett lättare sätt kunna visualisera det man har tänkt sig skapa / har skapat. Jag tror att mitt sätt att arbeta var väldigt likt den agila metoden då snabba förändringar kunde ske utan att behöva följa en specifik struktur. Arbetet planerades in i stilen "whenever i can, i will code" vilket innebär att när man har möjlighet så kodar man, detta skedde främst under kväll/natt tid. Jag valde att koda ett objektorienterad program i Java för att skapa spelet. Jag använde även mig av Git för versionshantering och commitade i princip varje gång jag kände mig nöjd med en uppgift och comittade igen vid modifiering. Allt för att underlätta en klar och läsbar log. Jag använde mig av designmönstret singleton för att säkerställa att det endast fanns en instans av spelet i gång samtidigt.

Utveckling
Jag började med att skapa en Main, meny klass, monster klass, hero klass och en subklass Warrior. Tanken var att göra det enklaste först, vilket var att få spelet att starta och avsluta genom en meny klass och while loop med switch i main. Därefter skapades en hero klass och en subklass warrior som var tänkt att ha så lite kod som möjligt i. Tanken var att endast modifieringar av Hero klassen som karaktäriserar en Warrior ska finnas i Warrior klassen. Detta för att inte behöva skriva dubbel kod i mina subklasser. När jag fick den att funka med alla modifieringar, passiva abilities, guld & xp så gjordes detta för en subklass Warlock.

I samband med att försöka få en fungerande hero & warrior klass skapades en monster klass. Dessa klasser hänvisades i main klassen som för det mesta bestod utav while loopar & switches. Spelet i sitt tidiga stadie började med att man endast kunde välja en hero (warrior) som slåss mot ett monster. Båda har ett default värde på hp och skada som de gör och hp dras av varje gång man tar skada. Vinner heron får man en ny lvl, fullt hp och mer skada. Dör heron så förlorar man spelet. Jag ville göra det så enkelt för mig som möjligt och upptäckte att ifall man definierar default värden på hp & skada så kan man använda lvl som en modifierare till dessa värden. D.v.s vid varje ökad lvl så ökar också hp & skada enligt en algoritm. Detta sparar väldigt mycket tid och energi då jag även hade en liten tanke på att skapa specifika värden för varje lvl. Dock hade jag i min kod att ett nytt objekt av både monster och valda hero klass skapades på nytt vid varje nivå, detta ändrades i ett senare skede då jag märkte att variabler som guld kommer resettas vid varje ny instansiering.

Spelet utvecklades sedan väldigt naturligt, så fort kampen var buggfri lades det till nya saker som t.ex en boss var femte nivå. Där använde jag mig av en modulo 5 algoritm för att trigga boss fighten. Likadant gjordes efter att ha skapat en shop men i stället att den skulle triggas var fjärde nivå genom mod 4. Spelet gick från ett automatiserat program där man bara får trycka på start till att kunna välja i menyn, välja hero,vilka attacker man ska utföra och köpa en vara.

Datatyper

• Integer (heltal): Används för att representera numeriska värden som karaktärens hälsa, attackstyrka och föremåls mängd som t.ex guld. Det fanns ingen användning för decimaler.

• Boolean (boolesk): Används för att representera sant/falskt-värden som om heron eller monstret är vid liv eller är döda. Den användes även för att hålla igång / avsluta while loopar.

• String (sträng): Används för att representera textdata som namn på karaktärer, föremål samt all typ utav textoutput i form av information, story och attacker.

• ArrayList: Används för att lagra en samling av värden av samma datatyp i en lista. ArrayList är smidig när man ska lägga till och ta bort innehåll som t.ex i mitt fall i en “inventory”.

• Object (objekt): Används för att representera en samling av relaterad data och funktioner, som t.ex. en karaktär eller ett monster. Objekt är användbart för att organisera komplexa datastrukturer och för att dela återanvändbar kod mellan olika delar av spelet.

Vem gjorde vad:

I vårt projekt fördelade vi arbetsuppgifterna på följande sätt:

Kerim: Kerim var ansvarig för design och implementering av i stort sett allt. Alla klasser, metoder, strukturer, brainstorming, loggföring och skapandet av klassdiagram.

ChatGPT: ChatGPT var ett stöd i detta projekt. Ain användes främst till att förklara varför vissa buggar uppkom samt gav sin åsikt gällande koden som skapades.

Vad kunde ha gjorts bättre:

Spelet är i helhet inte på en avancerad nivå. Det finns otroligt mycket utvecklingsmöjligheter. Nedan kommer några exempel.

* Weapons som förstörs när man slår på fienden. Nu har man alltid ett vapen som inte går sönder.

* Utveckla shop valen. Möjliggöra flera vapen som har olika attributer. Detta gör det även lämpligare att variera beroende på vilken hero subclass man har valt.

* Hp potions eller regen. Nu förlorar heron liv efter varje kamp utan att återfyllas. Det hade kunnat göras att den inte resettas och att man kan köpa potions i shoppen som läggs till i ens inventory. Man hade även kunnat skapa så att ens hp fylls på en viss andel t.ex varje runda samt som skalar med lvl och kan påverkas av ens items.

* Defense. Nu har defense ingen betydelse mer än printad text. Det vore intressant att lägga till en algoritm som beräknar att en viss andel av ens defense minskar monstrets attack och vice versa.

*Xp. En xp bar hade möjliggjort att heron samlar på sig xp efter varje strid tills den uppnår en gräns och då kan lvla upp. Xp baren ska då resetta samt kan skala upp och utöka sin gräns. Upplevlingen ska i sin tur utöka herons attributer.

* Mana & mana potions. Likt hp bör en spell caster class ha mana som den förbrukar vid special attacker. Denna mana ska även gå att fylla på.

* Flera monster samtidigt. Det vore väldigt intressant att möjliggöra att slumpa fram olika monster som man slåss mot samtidigt. Där man också får välja vem man får attackera.

* Gold funktion. Jag hade kunnat utveckla min goldfunktion till att ge en random summa gold efter varje monster, lite mindre för monster med lägre lvl och extra för boss.

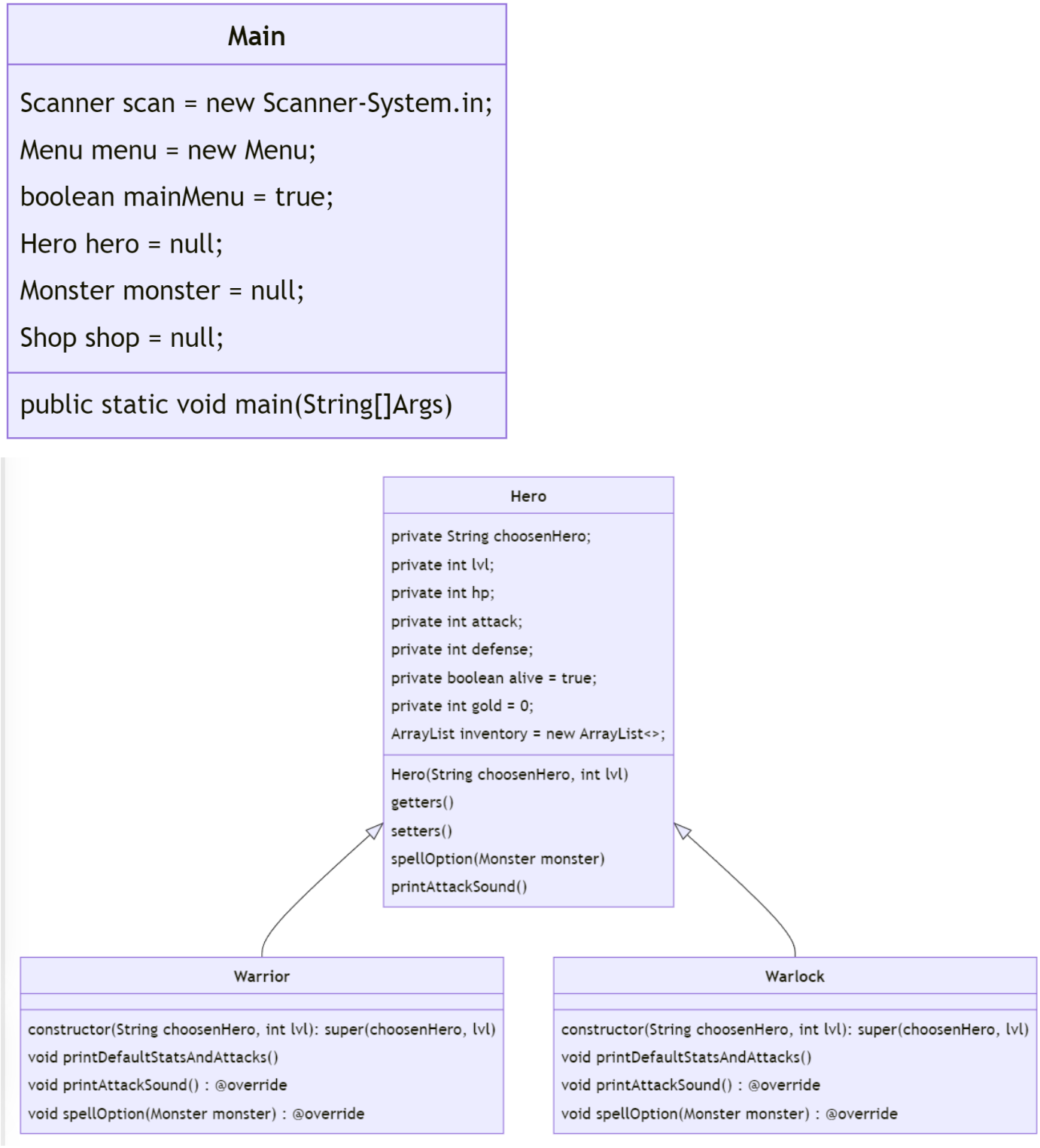
* Save game funktion. Det mest intressanta som är främmande för mig i nuläget hade varit att kunna spara det påbörjade spelet på en fil så att man inte behöver börja om varje gång man startar spelet.

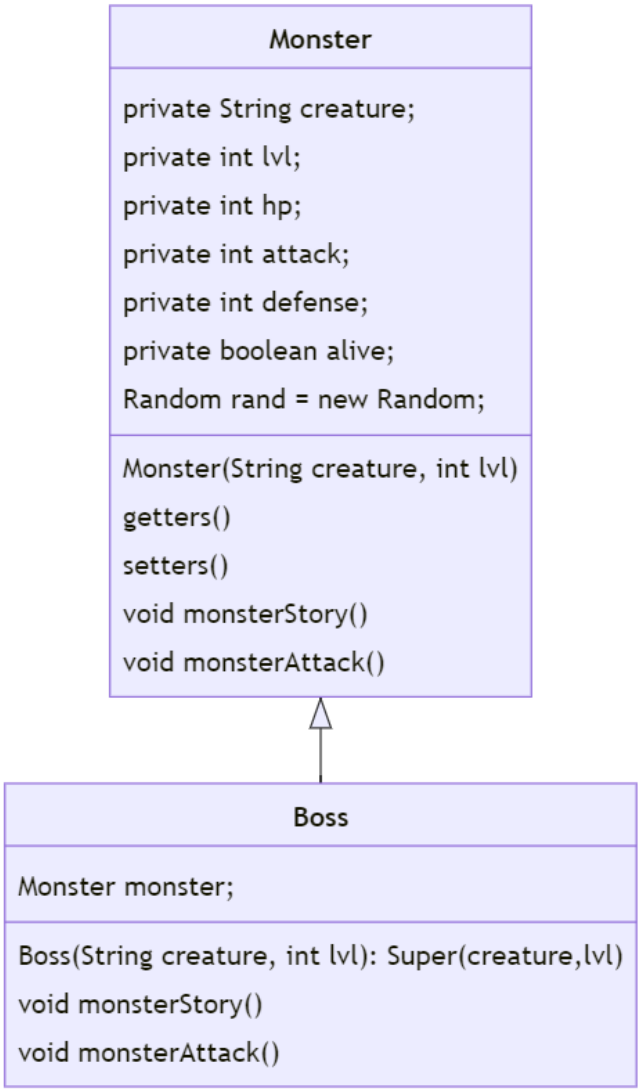
Hade där funnits mer tid och resurser, till exempel andra människor att bolla ideer med hade dessa bitar säkerligen gått att implementera. Jag hade dock även velat gå igenom mitt spel som jag har åstadkommit, metod för metod med någon senior programmerare då jag vill lära mig nu i mitt tidiga utvecklingsskede hur man skriver metoder på bästa sätt utifrån KISS, YAGNI e.t.c. Jag är generellt alltid tveksam till ifall jag skriver min kod på det bästa sättet. Det hade också varit väldigt roligt och lärorikt att få implementera allt man har lärt sig och saker man inte riktigt har fått använda som t.ex alla olika typer av listor, maps , trees, enum o.s.v. Även om det låter väldigt onödigt så hade det varit otroligt gynnsamt ur ett lärande perspektiv.

Slutsatser:
Jag är i helhet väldigt nöjd med detta spel. Anledningen till varför jag inte anser det vara på en avancerad nivå tror jag ligger i att allt hårda arbete där detta var helt nytt för mig gjordes till hangman spelet. Så för detta projekt stod jag redan på stadiga ben. Det som också har underlättat är att jag har kollat på väldig mycket videomaterial online som jag också har antecknat på datorn. Anteckningarna har blivit som en "referens" som jag hela tiden har kunnat vända mig till ifall jag blev osäker hur saker skulle kodas. Jag är väldigt glad att jag fick mindre motgångar för detta projekt då det har fått mig att tycka det är kul när jag kodar. Hade jag haft samma motgångar som med hangman spelet hade denna rapport haft ett helt

annat innehåll. Jag ser också väldigt mycket potential till att utveckla min kod vilket är roligt eftersom man kan fortsätta utmana sig själv. En typiskt utmaning och funderare för mig är när jag skriver en kod på ett visst sätt och vill modifiera den, finner ja ibland att den hade kunnat skrivas på ett helt annorlunda sätt. Jag märkte tydligt i detta projekt och nu i slutspurten att en minsta lilla modifiering kan få hela programmet att krasha, vilket har lärt mig att det ibland är bättre att köra på det man har, så länge det funkar än att tänka om.

Klassdiagram





Menu

Scanner scan = new Scanner-System.in;

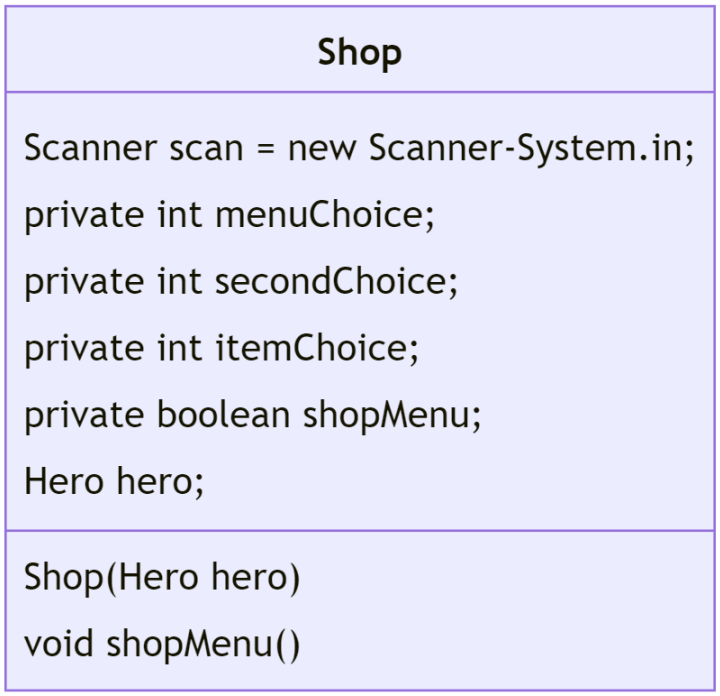
Warrior warrior = new Warrior;

Warlock warlock = new Warlock;

int choice;

int mainMenu()

int heroMenu()



Backlog

Övergripande

- Design och implementering av Monster-klassen och dess metoder
- Design och implementering av Boss-klassen och dess metoder
- Design och implementering av Hero-klassen och dess metoder
- Design och implementering av Warrior-klassen och dess metoder
- Design och implementering av Warlock-klassen och dess metoder
- Design och implementering av Menu-klassen och dess metoder
- Design och implementering av Shop-klassen och dess metoder
- Design och implementering av Shop-klassen och dess metoder
- Hantering av interaktionen mellan Hero & Monster samt Hero & Boss
- Testa spelet för att säkerställa att det fungerar som det ska och skapa dokumentation samt en rapportmall

Utförligare backlog

Menu klassen

- Skapa metoden mainMenu() som printar välkomsttext vid uppstart av spelet samt två val(starta/avsluta spelet) och frågar användaren om en input.
- Skapa metoden heroMenu() som printar en förklarande text samt alternativa val av heros och frågar användaren om en input.
- Uppdatera mainMenu() med en do while funktion samt try catch så att man försäkrar sig att spelaren enbart kommer vidare med att trycka 1 eller 2 samt undviks krashar.

Shop klassen

• Skapa switchar och printa alla möjliga köp alternativ. Köpen ska bestå utav föremål som har attributer och en kostnad. Vid lyckat köp ska kostnaden dras av från Heron's pengar. Vid misslyckat köp ska det printas att det fattas pengar. Varan ska också läggas i en inventory gjorts utav en arraylist.

Main klassen

• Skapa en while loop för huvudmenyn som håller spelet igång vid uppstart. Tills användaren väljer att avsluta spelet eller vinner spelet.
• Implementera en switch med case 1 för att starta själva spelet och 2 för att avsluta spelet. Värden får den av mainMenu() som ger en int i return.
• Skapa en switch vid start av spelet får ett int värde av heroMenuChoice(). Värdet leder sedan till att motsvarande objekt av en av subklasserna instansieras i Hero.
• En input efterfrågas av spelaren gällande hur många nivåer den vill spela och värdet sätts i en for loop som kör kills heron dör vilket innebär att man förlorar och kommer tillbaka till huvudmenyn. Klarar man alla nivåer har man vunnit spelet och kommer tillbaka till huvudmenyn.
• Var tredje runda implementeras Shop.shopMenu() som är en egen klass. Var femte implementeras Boss som är en egen subklass till Monster.
• Skapa en delay för en mer användarvänligt upplevelse

Monster klassen

• Skapa relevanta attributer och spara dem i variabler.
• Skapa relevanta setters & getters, d.v.s endast de som kommer användas.
• Skapa monsterStory() med en unik story & unika attacker i monsterAttack()

Boss klassen

• Klassen ska extenda Monster och ska i konstruktorn ha ett unikt namn och lvl som förs över till superklassen.
• Implementera monsterStory() med en unik story & unika attacker i monsterAttack() som overridar superklassens.

Hero klassen

• Skapa relevanta attributer och spara dem i variabler.
• Skapa relevanta setters & getters, d.v.s endast de som kommer användas
• Skapa en attacksound & spellOption

Warlock/Warrior klassen

• Klasserna ska extenda Hero och ska i konstruktorn ha ett unikt choosenHeroName och lvl som förs över till superklassen.
• Printa en unik story för varje hero subklass.
• Overrida superklassens attackSound() med ett unikt ljud för klassen

- Overrida superklassens med unika attacker i spellOption.