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

1. Introduktion till projektet..... 2
2. Ytterligare bakgrundsinformation..... 2
3. Milstolpar..... 2
4. Övriga implementationsförberedelser..... 4
5. Utveckling och samarbete..... 4
6. Implementationsbeskrivning..... 6
   6.1. Milstolpar..... 6
   6.2. Dokumentation för programstruktur, med UML-diagram..... 6
7. Användarmanual..... 7

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

Även om denna del inte ska lämnas in förrän projektet är klart, är det **viktigt att arbeta med den kontinuerligt** under projektets gång! Speciellt finns det några avsnitt där ni ska beskriva information som ni lätt kan glömma av när veckorna går (vilket flera tidigare studenter också har kommenterat).

Tänk på att ligga på lagom ambitionsnivå! En välskriven implementationsbeskrivning (avsnitt 6) hamnar normalt på **3-6 sidor** i det givna formatet och radavståndet, med ett par mindre UML-diagram och kanske ett par andra små illustrerande bilder vid behov. Hela projektrapporten (denna sista halva av dokumentet) behöver sällan mer än 10-12 sidor.

6. Implementationsbeskrivning

I det här avsnittet, och dess underavsnitt (6.x), beskriver ni olika aspekter av själva *implementationen*, under förutsättning att läsaren redan förstår vad *syftet* med projektet är (det har ju beskrivits tidigare).

Tänk er att någon ska vidareutveckla projektet, kanske genom att fixa eventuella buggar eller skapa utökningar. Då finns det en hel del som den personen kan behöva förstå så att man vet *var* funktionaliteten finns, *hur* den är uppdelad, och så vidare. Algoritmer och övergripande design passar också in i det här kapitlet.

Bilder, flödesdiagram, osv. är starkt rekommenderat!

Skapa gärna egna delkapitel för enskilda delar, om det underlättar. **Ta inte bort några rubriker!**

Även detta är en del av examinationen som visar att ni förstår vad ni gör!

6.1. Milstolpar

Ange för varje milstolpe om ni har genomfört den helt, delvis eller inte alls.

Detta är till för att labbhandledaren ska veta vilken funktionalitet man kan ”leta efter” i koden. Själva bedömningen beror *inte* på antalet milstolpar i sig, och inte heller på om man ”hann med” milstolparna eller inte!

6.2. Dokumentation för programstruktur, med UML-diagram

Programkod behöver dokumenteras för att man ska förstå hur den fungerar och hur allt hänger ihop. Vissa typer av dokumentation är direkt relaterad till ett enda fält, en enda metod eller en enda klass och placeras då lämpligast vid fältet, metoden eller klassen i en Javadoc-kommentar, *inte här*. Då är det både enklare att hitta dokumentationen och större chans att den faktiskt uppdateras när det sker ändringar. Annan dokumentation är mer övergripande och saknar en naturlig plats i koden. Då kan den placeras här. Det kan gälla till exempel:

- **Övergripande programstruktur**, t.ex. att man har implementerat ett spel som styrs av timer-tick n gånger per sekund där man vid varje sådant tick först tar hand om input och gör eventuella förflyttningar för objekt av typ X, Y och Z, därefter kontrollerar kollisioner vilket sker med hjälp av klass W, och till slut uppdaterar skärmen.

- **Översikter över relaterade klasser** och hur de hänger ihop.
    - Här kan det ofta vara bra att använda **UML-diagram** för att illustrera – det finns även i betygskraven. Fundera då först på vilka grupper av klasser det är ni vill beskriva, och skapa sedan ett UML-diagram för varje grupp av klasser.
    - Notera att det sällan är särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram (vad är det då man fokuserar på?). Hitta intressanta delstrukturer och visa dem. Ni behöver normalt inte ha med fält eller metoder i diagrammen.
    - **Skriv sedan en textbeskrivning av vad det är ni illustrerar med UML-diagrammet**. Texten är den huvudsakliga dokumentationen medan UML-diagrammet hjälper läsaren att förstå texten och få en översikt.
    - IDEA kan hjälpa till att göra klassdiagram som ni sedan kan klippa och klistra in i dokumentet. Högerklicka i en editor och välj Diagrams / Show Diagram. Ni kan sedan lägga till och ta bort klasser med högerklicksmenyn. Exportera till bildfil med högerklick / Export to File.

I det här avsnittet har ni också en möjlighet att visa upp era kunskapen genom att diskutera koden i objektorienterade termer. Ni kan till exempel diskutera hur ni använder och har nytta av (åtminstone en del av) objekt/klasser, konstruktorer, typhierarkier, interface, ärvning, overriding, abstrakta klasser, subtypspolymorfism, och inkapsling (accessnivåer).

Labbhandledaren och examinatorn kommer bland annat att använda dokumentationen i det här avsnittet för att förstå programmet vid bedömningen. Ni kan också tänka er att ni själva ska vidareutveckla projektet efter att en annan grupp har utvecklat grunden. Vad skulle ni själva vilja veta i det läget?

När ni pratar om klasser och metoder ska deras namn anges tydligt (inte bara ”vår timerklass” eller ”utritningsmetoden”).

Framhäv gärna det ni själva tycker är **bra/intressanta lösningar** eller annat som handledaren borde titta på vid den senare genomgången av programkoden.

Vi räknar med att de flesta projekt behöver runt **3-6 sidor** för det här avsnittet.

7. Användarmanual

När ni har implementerat ett program krävs det också en manual som förklarar hur programmet fungerar. Ni ska beskriva programmet tillräckligt mycket för att en labbhandledare själv ska kunna *starta det, testa det och förstå hur det används*.

Inkludera flera (**minst 3**) **skärmdumpar** som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument, inte i separata filer. Sikta på att visa de relevanta delarna av programmet för någon som *inte* startar det själv, utan bara läser manualen!

(Glöm inte att ta bort våra instruktioner, och exportera till PDF-format med korrekt namn enligt websidorna, innan ni skickar in!)