Projektbeskrivning

The legend of Hanry:
Adventure of användaren

2022-03-22

Projektmedlemmar:

Henry Andersson henan555@student.liu.se

Hannah Bahrehman hanba478@student.liu.se

Handledare:

Philip Rettig phire844@liu.se

Innehåll

Projektplan

Läs först:

https://www.ida.liu.se/~TDDD78/labs/2022/project/intro

https://www.ida.liu.se/~TDDD78/labs/2022/project/select

https://www.ida.liu.se/~TDDD78/labs/2022/project/documents

Ni väljer själva vilken sorts projekt ni vill utföra, men det finns vissa begränsningar som man behöver tänka på. Läs om detta på ovanstående sidor. Det är viktigt för att ni inte ska måla in er i ett hörn med ett projekt som inte är lämpligt för kursen.

Projektplanen skriver ni i samband med första inlämningen, gärna under tiden ni arbetar på sista labben för att göra det möjligt att få kommentarer innan projektstart. Små kompletteringar kan göras senare, men försök få med så mycket som möjligt redan från början.

Använd denna mall, i t.ex. LibreOffice eller Word -- inte en egentillverkad

Låt alla rubriker vara kvar, med samma **numrering** och **rubriktext**

Ta bort de övriga instruktionerna, till exempel denna text och ”Beskriv relativt

kortfattat…” nedan, när ni har gjort vad instruktionerna säger!

1. Introduktion till projektet

Beskriv relativt kortfattat det program ni tänker utveckla under projektet.

Tänk er att läsarna inte vet särskilt mycket om vad t.ex. Tower Defense är eller vad man kan ha en CPU-simulator till, och att ni ska förklara vad det går ut på. Beskriv på en nivå där läsarna kan förstå om de skulle vara intresserade av att ladda ner slutresultatet eller inte. Några stycken text kan vara lämpligt – gärna även en bild eller två.

Använder du ett inspirationsprojekt? I så fall:

Ange vilket inspirationsprojekt du har utgått från!

Det är tillåtet att kopiera text därifrån.

Actionäventyr spel

Inspirerat av *The legend of Zelda*

En dungeon med:

Olika rum

Pussel

Monster

Bossmonster

2. Ytterligare bakgrundsinformation

För vissa typer av program kan det finnas en hel del bakgrundsinformation som inte behövs för att förstå ungefär vad projektet går ut på men som däremot behövs för själva implementationen. Den skriver ni ner i det här avsnittet. Skriver ni en FTP-klient behöver man känna till FTP-protokollet; skriver ni ett brädspel behöver man veta samtliga regler för spelet.

Ni får gärna peka på information på nätet med en URL om informationen är omfattande, men åtminstone en kortare sammanfattning (några stycken) ska ändå finnas här i dokumentet.

Använder du ett inspirationsprojekt är det tillåtet att kopiera text därifrån.

3. Milstolpar

Tänk efter **hur ni kan utveckla programmet stegvis** så att ni hela tiden kan testa koden, verifiera att den fungerar, och se konkreta resultat! Programmering är roligt, men att verkligen

se ett fungerande program är ännu roligare.

Konkretisera stegen genom att dela upp projektet i en sekvens **milstolpar,** där varje milstolpe anger en viss funktionalitet som ska finnas i projektet. Tanken är att ni anger ett antal sådana milstolpar under planeringen och därefter använder dem som ledning för er själva under implementationsfasen. Varje steg bygger vidare på de tidigare stegen med ny funktionalitet. Efter varje steg, inklusive det första, ska det finnas en testbar ”produkt” som har någon form av meningsfull funktionalitet. **Detta är något som många tidigare studenter säger sig ha haft mycket nytta av!** Exempel ges nedan.

Det är väldigt svårt att i förväg veta exakt hur mycket som är rimligt att implementera under kursens gång. Därför vill vi att ni anger ett antal milstolpar där ni tror att de inledande **med lätthet kommer att hinnas med**, medan de mot slutet av listan kommer att **ta betydligt mer tid** än ni har under kursen. För de flesta projekt bör det gå relativt lätt att dela upp funktionaliteten i minst **20-25** steg, eller varför inte **40-50**?

Listan är alltså **inte en kravlista** där någon kommer att klaga om ni inte uppnår allt, utan är helt och hållet tänkt att vara en vägledning till er själva (och ge labbhandledarna en möjlighet att kommentera och ge förslag, så klart). Genom att tänka genom stegen i förväg, och även ta med sådant som ni troligen inte hinner med, får ni ett bättre underlag när ni ska designa grunden i ert projekt. Att veta att man kanske någon gång i sitt spel ska implementera nätverksstöd eller låta flera användare spela på samma gång kan till exempel påverka även er första uppbyggnad av spelarobjekt och liknande. Dessutom har ni garanterat att ni har ett sätt att gå vidare om projektet visade sig vara enklare än ni trodde, så ni behöver mer funktionalitet för att få tillräcklig omfattning.

Stegen i listan bör till största delen bestå av **mätbara krav** (även om det alltså inte är någon som kommer att titta vilka ”krav” som har uppfyllts). Med andra ord, man bör kunna svara objektivt "ja" eller "nej" på om ett steg är uppnått eller inte, vilket man till exempel kan för "Spelet ska kunna styras både med tangentbord och med mus". Om ni skriver att "spelet ska vara snyggt" är det svårare att bedöma – men eftersom det inte är en ren kravlista kan ni t.ex. skriva ner att ”I detta steg lägger vi lite mer tid på spelets utseende”.

Skriv ner milstolparna genom att bygga ut tabellen nedan. Exempel ges för Tetris-spelet.

Använder ni ett inspirationsprojekt? I många fall är de angivna milstolparna i beskrivningen väldigt preliminära och ungefärliga. Ni behöver *vidareutveckla* dem för att beskriva i mer detalj vad som ska göras och hur just *ni* kommer att realisera dem. Ni kan också vilja ta bort, lägga till eller ändra milstolpar.

#	Beskrivning
1	Kan visa en ruta på skärmen (spelplanen).

	(Det finns datatyper för spelplan och ”kvadrater” samt funktionalitet för att slumpa fram spelplaner med 50% kvadrater och resten av positionerna tomma. Det finns också en grafisk komponent som kan visa en spelplan i Tetris standardfärger. En testklass kan skapa en slumpmässig spelplan och visa den grafiskt i ett fönster. Det går att stänga fönstret genom att klicka på stängknappen.)
2	Laddat in bilder (sprites). Kan visa bilder på skärmen. (Samtliga bricktyper är implementerade. Den grafiska komponenten kan nu visa både en spelplan och en bricka som placeras ”över” spelplanen på givna x/y-koordinater. Testklassen slumpar även fram en bricktyp och placerar den på slumpmässiga koordinater.)
3	Få en bild att röra på sig. Bilden ska röra på sig med instruktioner från tangentbordet.
4	Kollision kan detekteras.
5	Kan byta rum. När en bild/gubbe är på en specifik plats (dörr eller liknande öppning) kommer ett annat rum visas.
6	Implementerat fiender och enkel rörlighet.
7	Implementerat enkelt hälsa-system för fiender och huvudkaraktären.
8	Skapat grundläggande GUI.
9	Karaktärer kan skada och bli skadad (utökat kollision). Huvudkaraktären skadas genom att gå in i fienden.
10	Utökat hälsa-systemet.
11	Utökat spelplanen. Det finns väggar, golv och dörrar.
12	Implementerat inventory.
13	Implementerat svärd
14	Implementerat fler vapen.
15	Kan skada fienden med vapen.
16	Skapa dörrar med nycklar
17	Skapa boss
18	Skapa pussel / utöka spelplanen
19	Skapa slutskärm/slutskärm

20	Spelet är klart :^).
21	
22	
23	
...	

4. Övriga implementationsförberedelser

Man kan ibland spara mycket tid på att först ta en ordentlig funderare på t.ex. vad programmet ska göra, ungefär hur funktionaliteten kan delas upp mellan olika klasser, osv.

Tänk därför gärna genom den grundläggande strukturen på programmet och skriv ner de tankar ni har om detta. Det går inte nödvändigtvis att bestämma hela strukturen i förväg, men vissa delar kanske kan planeras i detalj och vissa delar kan beskrivas i stora drag. Detta avsnitt är inte del av examinationen, men ju mer ni kan beskriva här, desto större chans att ni får användbara kommentarer från handledaren.

Många av tidigare års studenter har rapporterat att de hade stor nytta av det här.

5. Utveckling och samarbete

Detta gäller främst för grupper med mer än en medlem, men flera punkter är även bra att fundera på för enskilda.

Utöver att beskriva själva projektet behöver ni också diskutera *projektarbetet* och komma överens om hur ni tänker gå tillväga. I vissa kurser görs detta genom fullständiga gruppkontrakt som signeras av medlemmarna. Här går vi inte riktigt så långt, men det finns ändå en stor poäng med att se till att eventuella skillnader i gruppmedlemmarnas inställning uppdagas tidigt, istället för att man upptäcker missförstånd när det närmar sig deadline.

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?

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?

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?

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

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?

Det kan också vara bra att diskutera hur ni rent allmänt känner inför att genomföra projektet. Roligt? Svårt? Något annat?

Detta är absolut inte betygsgrundande, utan ska vara en hjälp till er själva. Ni behöver inte heller fylla i svaren *här* om ni föredrar att undvika det, men ni bör definitivt skriva ner svar för er själva så att det inte finns någon tvekan om vad ni har diskuterat. Det kanske känns som att det inte behövs eftersom ni är helt överens – men det är bara då man *kan* göra det. Om ni väl skulle få motsättningar i gruppen är det redan för sent...

(Resten av dokumentet ska inte lämnas in förrän projektet är klart, men titta ändå genom allt för att se vilka delar ni behöver arbeta med och fylla i *kontinuerligt* under projektets gång!)

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!)