Projektbeskrivning

<Plattformsspel>

2015-04-15

Projektmedlemmar:

Jens Reimers jenre974@student.liu.se

Yousif Touma youto814@student.liu.se

Handledare:
Mikael Nilsson mikael.a.nilsson@liu.se

1. Introduktion till projektet

Vårt projekt ska bli ett plattformsspel. Det går ut på att du har en karaktär som kan röra sig inuti en spelplan i 2D. Karaktären ska kunna hoppa samt röra sig i sidled. Under spelets gång så samlar man poäng i form av grafiska objekt. Det finns även fiender att undvika att ta skada av.

2. Ytterligare bakgrundsinformation

3. Milstolpar

1 – Ta reda på hur och implementera sprites och utritning av dem

2 Skapa en ritbar klass (en entity)

3 skapa en acceleration och velocityentity

4 – kollisionshantering

5 – Skapa en spelbar spelare

5a – Rita ut en spelare

5b – tangentbordsstyrning för vågrät rörelse

5c – gravitation och hoppsystem

6 – Skapa fiender

7 – Poäng (typ mynt)

8 – game over

9 – animation

10 ljud

11 – livsystem

12 – powerups

13 – fler fiendetyper

14 – svårighetsgrad ökar med tid, highscore, olika spelarkaraktärer

4. Övriga implementationsförberedelser

Vi har en form av arvsträd där vi högst upp har Entity som ska ta in en sprite och en vektorposition eller bara koordinater. Sedan har vi en CollisionEntity som ska ärva denna och här ingår alla objekt som ska hantera kollision. Härifrån har vi poäng(mynt) som en grej, wall (som även innefattar plattformar) som en gren och sedan velocity och accelerationentity. Härifrån finns sedan spelare och fiender som ska både röra sig och accelerera utöver att kollidera.

4. Utveckling och samarbete

Vi hade till en början tänkt jobba mycket tillsammans för att båda ska hänga med på strukturen.
Ju längre vi kommer i projektet så kommer vi kunna jobba med att implementera funktionalitet på egen hand när båda har koll på hur strukturen fungerar. För att versionshantera har vi satt upp ett repo på IDAs gitlab. Vi ska försöka kommentera all ny funktionalitet, speciellt när vi gjort något på egen hand så man kan sätta sig in i koden.

Vi vill även köra en implementationsfreeze med ca 1 1⁄2 vecka kvar för att finslipa koden och få allt att fungera som det ska.
Vi har båda även kommit överens om våra betygsambitioner.

Slutinlämning

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).

5. Implementationsbeskrivning

Den här sektionen använder ni för att beskriva hur projektet är strukturerat och implementerat. Algoritmer och övergripande design passar också in i det här kapitlet (bilder, flödesdiagram, osv. kan vara användbart). Skapa gärna egna delkapitel för enskilda delar, om det underlättar.

5.1. 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!)

5.2. Dokumentation för programkod, inklusive 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 placeras då 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 (det är sällan särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram).
    - 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.

Labbhandledaren och examinatorn kommer bland annat att använda den här dokumentationen 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? Om viktig information saknas kan ni få komplettera.

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 lösningar eller annat som handledaren borde titta på vid bedömningen.

Se även sidan om kvalitetskriterier på webben.

Vi räknar med att de flesta projekt behöver runt 2-5 sidor för det här avsnittet.

5.3. Användning av fritt material

Vi använder ett paket som heter Libgdx, som bidrar med framförallt den grafiska delen men kan även användas för resurshantering, tangentbordsstyrning samt för ljudeffekter. Vi har använt biblioteket för att hantera grafik, tangentbordsstyrning i spelet samt resurshantering.

5.4. Användning av objektorientering

För varje betygsnivå måste ni peka ut hur ni har använt ett visst antal olika **typiska egenskaper hos just objektorienterade språk.** Se projektsidorna på webben för detaljerade instruktioner!

1. …
2. …
3. ...

5.5. Motiverade designbeslut med alternativ

1. 2015-03-12
   Enumklasser
   Vi har valt att använda enumklasser för att beteckna olika typer, i vårt fall exempelvis olika sidor av hitboxes samt spelobjekt. Anledningen till det är att vi vill kolla med vad för objekt vi kolliderar med samt från vilken sida så vi vet var vi ska placera t.ex. spelaren när han krockar med en vägg till skillnad från om han krockar med ett mynt. Som man kunde gjort annars var att ta in ett heltal i metoder där vi istället tar in en enumtyp och göra en switchsats med case 1, 2, 3 osv men det är väldigt otydligt vad som görs i en sådan sats. Därför har vi istället if-satser eller switch-satser med de olika möjliga fallen. Detta ökar läsbarheten markant.
   För exempel se enumklasser GameObject, Side samt metod seperateSide i MovableEntity eller onObjectDeath i Game.
2. 2015-03-12
   Abstrakta klasser
   Vi har valt att använda abstrakta klasser för implementationsärvning. Anledningen till det är att vi vill ha kod som används av flera klasser utan att behöva instansiera ett objekt av just den klassen, utan bara dess subklasser. Vi hade istället kunnat använda gränssnitt för att ha klasser med gemensam funktionalitet men eftersom många, om inte alla metoder för vissa klasser ser likadana ut har vi hellre abstrakta klasser med abstrakta metoder där det behövs olika implementation. För exempel se abstrakta klasser Entity, MovableEntity, CollisionEntity samt konkreta klasser player, enemy, wall. Exempel på metoder med olika implementation är moveLeft och doAction som finns som abstrakta metoder hos MovableEntity

5.6. Bredd inom Java och objektorientering

För varje betygsnivå måste ni använda ett visst antal specifika OO- eller Java-finesser. Här ska ni beskriva vilka ni har använt och var. Var tydliga – ange klassnamn och metodnamn där det är lämpligt. Se projektsidorna på webben för detaljerade instruktioner!

1. …
2. …
3. …
4. …
5. …
6. …
7. …
8. ...

6. 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 fungerar.

Inkludera flera (minst 3) skärmdumpar som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument.

Starta spelet genom genvägen som skriker om att startas. Den ligger på toppnivå i projektmappen.

När du kommer in i spelet kommer du först komma till en meny som ber dig välja vilken karta du vill spela på. Efter att du har valt karta kommer spelet att starta och du spawnar. Du styr din karaktär med piltangenter höger, vänster samt upp för hopp. Spelet går ut på att undvika fiender (eller döda dem genom att hoppa på dem) samt samla poäng.

7. Slutgiltiga betygsambitioner

Ange här vilket betyg ni slutgiltigt siktar på i ert projekt. Se sedan till att ni har följt de kvantitativa kraven för detta betyg och att detta finns dokumenterat i projektbeskrivningen ovan.

8. Utvärdering och erfarenheter

Detta avsnitt är en väldigt viktig del av projektspecifikationen. Här ska ni tänka tillbaka och utvärdera projektet (något som man alltid bör göra efter ett projekt). Som en hjälp på vägen kan ni utgå från följande frågeställningar (som ni gärna får lämna kvar i texten så att vi lättare kan sammanställa information om specifika frågor):

- *Vad gick bra? Mindre bra?*
- *Lade ni ned för mycket/lite tid?*
- *Var arbetsfördelningen jämn? Om inte: Vad hade ni kunnat göra för att förbättra den?*
- *Har ni haft någon nytta av projektbeskrivningen? Vad har varit mest användbart med den? Minst?*
- *Har arbetet fungerat som ni tänkt er? Har ni följt "arbetsmetodiken"? Något som skiljer*

sig? Till det bättre? Till det sämre?

- *Vad har varit mest problematiskt, om man utesluter den programmeringstekniska delen? Alltså saker runt omkring, som att hitta ledig tid eller plats att vara på.*
- *Vad har ni lärt er så här långt som kan vara bra att ta med till era egna kommande kurser/projekt?*
- *Vilka tips skulle ni vilja ge till studenter i nästa års kurs?*
- *Har ni saknat något i kursen som hade underlättat projektet?*

Vi använder detta för att utveckla och förbättra kursen till nästa år. Vissa delar som är användbara som tips till andra studenter kan komma att citeras (givetvis helt anonymt!) under föreläsningar eller på websidor.

(Glöm inte att exportera till PDF-format innan ni skickar in!)