LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Kodgranskningsprotokoll

Författare

Alexander Jonsson, alejo720@student.liu.se
Joakim Johansson, joajo229@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2016
Version 1.1

2016-12-09

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Lagt till mottagen feedback	2016-11-09
1.0	Första version, positiv och konstruktiv feedback	2016-11-08

2 Kodgranskningsmötet

Datum: 2016 - 12 - 08

Given feedback från:
alejo720 (Alexander Jonsson)
joajo229 (Joakim Johansson)

Mottagen feedback från:
Sokrates Lamprou (andla830@student.liu.se)
Benjamin Fischer (benfi309@student.liu.se)

Mötet gick till på så sätt att vi gick igenom och förklarade varandras kod och diskuterade lite om vad som var bra och mindre bra med koden. Efter mötet pushade vi upp koden till git för att kunna inspektera vidare.

3 Vår granskning

Feedback till projektgruppen.

3.1 Positiv feedback

Saker som är bra och inte borde ändras.

3.1.1 Klassdiagram

Bra att ni har...

- Separation mellan medlemsvariabler och funktioner.
- En 'Engine'-klass som håller reda på spelet.
- Datatyp på alla variabler.
- Visat vilka klasser som äger andra klasser.

3.1.2 Koden

Bra att ni har...

- Påbörjat utveckling och testning av era klasser.
- Delat upp klasserna i filer.
- Gömt undan privata variabler inuti klassen.
- Skapat klasser med en tydlig uppgift i tanke.

I övrigt fina bilder och bra speltitel.

3.2 Konstruktiv feedback

Saker som borde tänkas en extra gång på.

3.2.1 Klassdiagram

Tänk på att...

- Ni har inte angett vilka funktioner/variabler som är privata/protected (görs med '-' respektive '#' framför).
- 'Angle' i tank tycker vi borde vara en double.
- Specificera klassen exakt. Byt ut t.ex. 'Event' mot 'sf::Event' och 'Sprite' till 'sf::Sprite'. Annars kan den som läser klassdiagram tro att det är en klass ni själva har implemterat.
- Ni behöver multiplicitet i erat klassdiagram. Det visar hur många instanser av en klass en klass äger. T.ex. en 'Tank' har 1 'Pipe', en 'Engine' har 2 'Tanks'. (* betyder ospecificerat antal).
- Publika funktionen 'Destroy' i projektilklassen har inte konsekvent namngivning eftersom den börjar med stor bokstav till skillnad från era andra publika funktioner.
- Er 'key_stream'-variabel är av typen 'sf::Event' men namnet får en att tro att det är specifikt ett 'Key Event'.
- 'Engine'-klassen är inte uppbyggd så det fungerar enligt följande krav:
  int main()
  {
  Engine engine;
  engine.run();
  return 0;
  }
  Dessutom har ni en 'Update()'-funktion i engine men tanken är ju att spel-loopen ska finnas i en run()-funktion som automatiskt uppdaterar spelet varje iteration.
- Raka pilar skulle se snyggt ut.
- change_angle(direction::bool) och change_power(amount:bool) har otydlig parametervariabel.

3.3 Koden

Tänk på att...

- Sätt 'const' på medlemsfunktioner som inte ska ändra på några variabler, t.ex. 'getters'. Sätt även 'const-reference(const &)' på avancerade datastrukturer i parametrarna där ni inte ska ändra på parametern.
- Fler förklarande kommentar, mer dokumentation.
- Ni behöver en gemensam basklass för alla 'sprites'. Ni skulle kunna ha en 'Enitity' klass som både 'Projectile' och 'Tank' ärver ifrån. Den klassen skulle till exempel kunna ha en funktion 'move' som 'Projectile'-klassen och 'Tank'-klassen ärver ifrån.
- För att minska beroendet mellan det grafiska och det logiska skulle ni kunna byta ut er 'sf::Sprite' objekt mot ett filnamn och ha spelarspriten i er grafiska klass. Tänk er att klasserna ni bygger borde kunna användas med endast terminalen som det grafiska.
- Ni har många publika medlemsfunktioner där ni skriver klassens namn tillsammans med vad funktionen gör. Detta är redundant då funktioner endast kan anropas ifrån objektet skapat av klassen. 'player.move' är minst lika tydligt som 'player.movePlayer'.
- Som vi nämnde i feedbacken om klassdiagrammet behöver ni strukturera 'Engine' klassen så ni kan köra erat spel på liknande vis:
  int main() {
  Engine engine;
  engine.run();
  return 0;
  }
- Ni borde ha en mapp med alla bilder så blir det lättare att navigera i projektet samt lättare att länka till bilder i mappen. Det gör det lättare om ni t.ex. vill implementera olika pansarvagnar med olika bilder.
- Er brist på kommentarer gör koden ibland svår att förstå.
- Era sprites i 'Pipe'-klassen heter 'piSprite' vilket tog ett tag för oss att förstå att det var förkortning för 'pipeSprite'. Det är viktigt med tydliga variabelnamn.
- Just nu representerar ni 'Player' och 'Pipe'-klassen genom en 'sf::Sprite'. Tänk om man vill använda sig av de klasserna på ett annan grafiskt gränssnitt. Just nu tänker ni: "En sprite är en Player". Tänk istället: "En Player har en 'Sprite'". Att ni har filväg i konstruktorn försämrar även klassens flexibilitet då någon kanske vill använda sig av Player klassen utan en Sprite. Generella klasser är att föredra.
- Ni skickar en 'char' för att välja vilken riktning ett objekt ska röra sig. Om ni istället skulle ha en 'enumeration' som representerar en riktning blir det lättare för användaren av koden att förstå vad den ska skicka in till funktionen. Som ni har nu krävs det kommenterar vilken 'char' som leder till vilken riktning.

4 Mottagen feedback

4.1 Feedback

1. Ni har ingen 'Game'-klass.
2. Variabeln 'win' i 'Player' är publik vilket inte en variabel bör vara.
3. Klassen använder sig av datatypen 'Point' men ni har inte inkluderat den i filen. Går det att kompilera ändå?
4. Ni har alla spelarsprites i 'GUI_Gameboard'-klassen. Vi tycker det skulle vara bättre att ha så att varje objekt håller i sin egen 'sf::Sprite' för att koncentrera allt som har med en sak att göra till en och samma klass. Då skulle ni även kunna använda er av SFML's inbyggda 'setPosition()' för att ändra positionen på ett objekt. Denna funktion tar alltså in en sf::Vector2f innehållandes två punkter.
5. Ni sätter position på en spelare i y-led men ni säger att en spelare endast kan sättas på x-led?
6. En 'GUI' är en metod för att underlätta interaktion med en människa och en dator, typiskt genom bilder och andra grafiska element. T.ex. menyer,HP-bars, ammunition och sikte för vapen (se exempelbild: https://postimg.org/image/6zdd0o901/). Därför tycker vi inte att ni borde kalla klasser som har att göra med själva världen för något annat. T.ex. 'GUI_Gameboard' håller spelararnas sprite vilket gör att den inte egentligen är en 'GUI' utifrån den tolkning vi har av vad en 'GUI' är i dess sanna bemärkelse.
7. I 'GUI'-klassen har ni variabel för ljud. Detta tycker vi inte tillhör det grafiska. Ni skulle kunna möjligtvis kunna lägga ljudet i en t.ex. spelarklassen och koppla det till en 'Action'.

4.2 Konsekvenser

1. Detta ledde till att vi fick bygga en 'Game'-klass istället för att köra allt i 'main'-funktionen.
2. Vi tog bort variabeln och bestämde oss för att hantera vilken spelare som vunnit genom att när det behövs titta vilka spelare som är döda.
3. Anledningen att det fungerar ändå är för att 'Entity'-klassen som 'Player'-klassen ärver ifrån inkluderar datatypen 'Point'. Men vi inkluderade 'Point' i 'Player'-klassen ändå för tydlighetens skull. Vi kollade även om vi hade missat samma sak på andra ställen i koden.
4. Vi vill bygga upp vår spelarklass på så sätt att den är så lite beroende av ett specifikt grafiskt gränssnitt som möjligt. Därför vill vi varken ha 'sf::Sprite' eller 'sf::Vector' i vår spelarklass. Därför ändrade vi inte, trots feedback.
5. Detta är för att en spelare måste starta på en y-position. Dessutom ger det flexibilitet om vi vill lägga till rörelse i y-led i framtiden. Därför ändrade vi inte, trots feedback.
6. Detta ledde till att vi tog bort 'GUI' framför de grafiska klassernas namn. Det ledde också till att vi tänkte en extra gång på vad de grafiska klasserna egentligen borde heta utifrån vad de ska representera.
7. Detta ledde till att vi flyttade ut ljudet från 'GUI' till logiska klasser. T.ex. Musiken ladde vi in i 'Level'-klassen.