LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Daniel Huber, danhu849@student.liu.se
Viktor Rösler, vikro653@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2019
Version 1.1

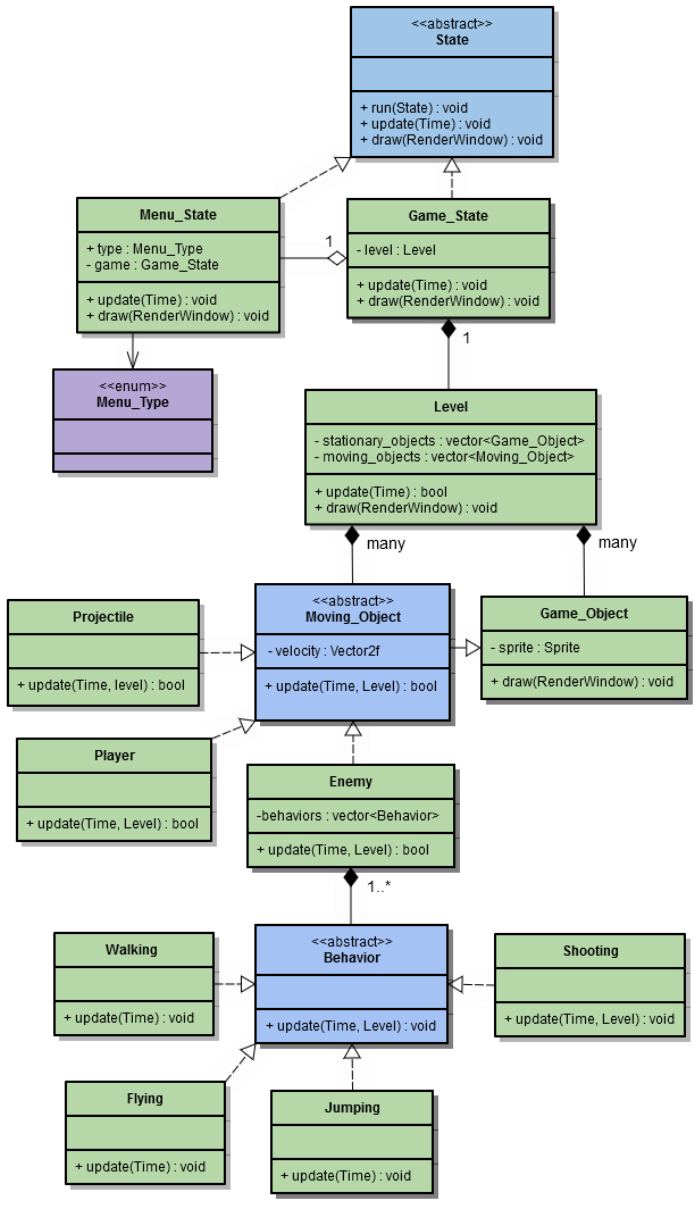
2020-11-25

Innehåll

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Klassdiagram och designdiskussion	201125
1.0	Designspecifikation 1:a utkast	201125

2 Klassdiagram



Figur 1: UML

3 Designdiskussion

Klassen State och dess underklasser Game_state och Menu_State representerar tillståndet spelet befinner sig i. Endast ett state är aktivt åt gången; Game_State om spelet är igång, eller Menu_State om spelaren befinner sig i en meny. Ett Menu_State innehåller ett Game_State som ritas ut bakom menyn. Anledningen till det är att vi vill få till en snygg övergång mellan Menu_State och Game_State. När spelet övergår från Menu_State till Game_State försvinner menyn från skärmen, och nivån som ritades ut bakom menyn blir spelbar. En nackdel med att ha ett Game_State i Menu_State är att spelet kan behöva att läsa in fler nivåer än de som spelas. T.ex. om spelaren bläddrar i nivåmenyn ska den nivå som är markerad visas i bakgrunden, men det är inte säkert att spelaren väljer att spela den nivån. Det vore lämpligt om vi endast behöver läsa in varje nivå en gång, oavsett hur många gånger den ska användas. Det skulle kunna genomföras med en singleton-klass som har ansvaret att läsa in och komma ihåg nivåer.

Menu_State kan representera olika sorters menyer, t.ex. en startmeny eller en pausmeny. Vilken sorts meny ett Menu_State objekt representerar bestäms när objektet skapas. För att dölja implementationen av Menu_State från andra klasser, och få till bra inkapsling, anropas konstruktorn till Game_Menu med en uppräkningstyp (enumeration) som argument. Uppräkningstypen bestämmer vilken typ av meny som ska skapas.

Klassen Level representerar en nivå i spelet. Klassen innehåller två vektorer med de objekt nivån består av. Uppdelningen finns för att vi vill endast anropa en update-funktion på de objekt som behöver uppdateras under spelets gång.

Varje Game_Object ritas ut genom att RenderWindow.draw() anropas med Game_Object objektets Sprite som argument. Det fungerar för att Game_Object ärver från sfml-klassen Drawable. Det gör att alla underklasser till Game_Objekt också kommer kunna ritas ut.

update() anropas i gameloopen i Game_State som i sin tur anropar update() i Level osv. Varje objekt ska uppdatera sig självt och sedan skicka anropet vidare nedåt. varje klass ska känna till så lite som möjligt om de andra klasserna i kedjan, men Level behöver skicka sig själv vidare så att objekt av typen Moving_Object kan veta om de kolliderar med andra objekt. spelaren/ vissa fiender behöver också känna till Level för att kunna skjuta (lägga till objekt av typen Projectile i en Level:s vektor med Moving_Object). update() till Moving_Object returnerar en bool så att Level ska kunna avgöra om de ska tas bort.

Ett objekt av typen Enemy består av ett antal beteenden. Varje beteende är ett objekt av någon undertyp till den abstrakta klassen Behavior. När funktionen update anropas på ett Enemy-objekt uppdateras objektet enligt de beteenden just denna fienden har. Poängen med att bryta ut fiendernas beteenden från Enemy-klassen är att det ska vara lätt att utöka spelet med fler fiendetyper. Vi har bossfiender som ett bör-krav till projektet. Det kravet ska kunna genomföras genom att vi återanvänder, och skapar nya, beteenden till den redan existerande Enemy-klassen.

4 Detaljbeskrivning av klassen Player

5 Detaljbeskrivning av klassen ...

6 Externa Filformat