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 1.2

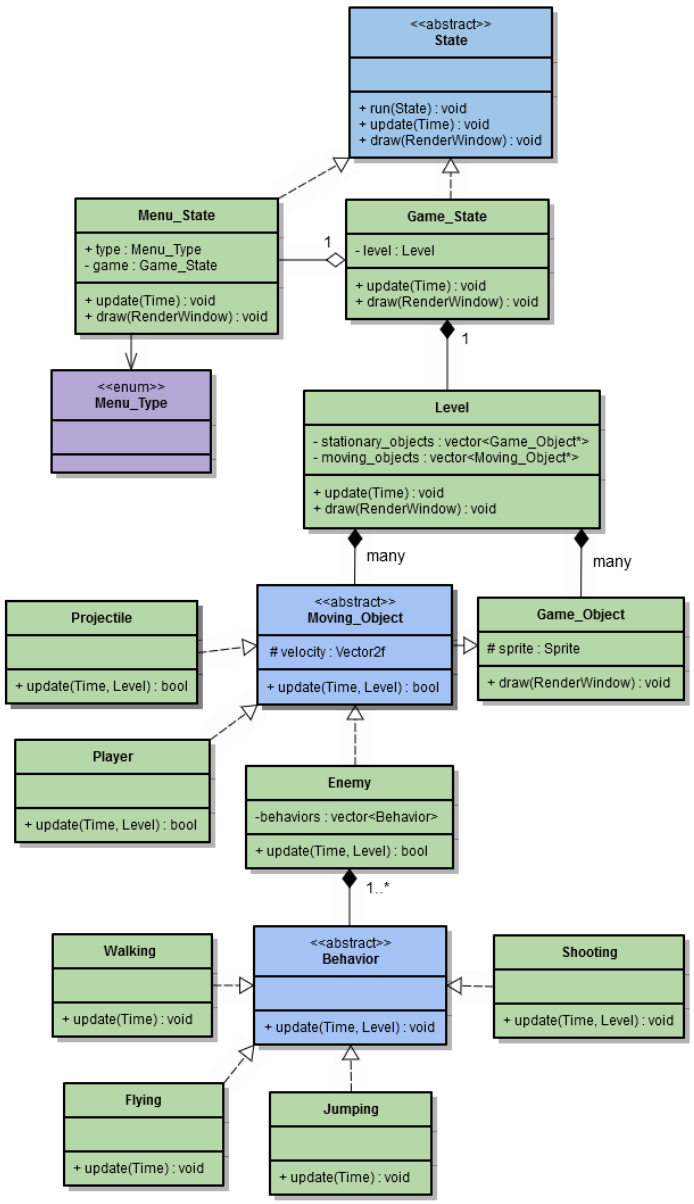
2020-11-27

Innehåll

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.2	Detaljbeskrivning och externa filformat	201127
1.1	Klassdiagram och designdiskussion	201125
1.0	Designspecifikation 1:a utkast	201125

2 Klassdiagram



Figur 1: Klassdiagram

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. när 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 högst en gång, oavsett hur många gånger den ska användas. Vi har gjort avvägningen att läsa in nivåer och behålla dem i minnet är mer effektivt än att behöva läsa in respektive nivå varje gång den väljs av användaren. Det skulle kunna genomföras med en singleton-klass som endast har ansvaret att läsa in och komma ihåg nivåer.

`Menu_State` kan representera olika sorters menyer, t.ex. en startmeny, en pausmeny eller en meny för inställningar. 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, objekt som ändrar nivåkoordinater, `Moving_Object` och objekt som har statiska nivåkoordinater, `Game_Object`. Uppdelningen finns för att endast behöva anropa en update-funktion på de objekt som behöver uppdateras under spelets gång och för att kunna uppdatera objekt av typen `Moving_Object` ovanför spelfönstrets vy.

Varje `Game_Object` ritas ut genom att `draw` funktionen i `RenderWindow` klassen anropas med `Game_Object` objektets `Sprite` medlem som argument. Det fungerar för att `Game_Object` ärver från sfml-klassen `Drawable`. Det gör att alla underklasser till `Game_Object` också kan ritas ut. En nackdel är att `Game_Objekt` blir större än nödvändigt, i form av den extra funktionalitet som finns i `Sprite` klassen. T.ex. har `Sprite` klassen en funktion för att flytta på sig, men ett objekt av typen `Game_Object` behöver inte kunna göra det.

Funktionen `update` anropas i gameloopen i `Game_State` som i sin tur anropar `update` i `Level` ner till `update`-funktionerna i varje `Behavior`. Varje objekt ska uppdatera sig själv och skicka anropet vidare nedåt. De `Game_Object` varje klass behöver 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 i nivån eller inte. `update` funktionen i `Moving_Object` och dess underklasser returnerar en bool för att `Level` ska kunna avgöra om objektet ska tas bort eller inte.

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 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 utan att behöva skapa en ny klass för bossar.

4 Detaljbeskrivning av klassen Player

Klassen `Player` representerar en figur i en nivå som en spelare kan styra. Det finns ett `Player` objekt per spelare i varje nivå, och varje `Player` objekt skapas och förtstörs tillsammans med nivån den är en del av. `Player` ärver från den abstrakta klassen `Movable_Objekt` som representerar alla typer av objekt i en nivå som kan flytta på sig. `Movable_Objekt` ärver i sin tur från klassen `Game_Objekt`, vilket är basklassen för alla typer av objekt i en nivå som har kollisionshantering.

4.1 Medlemmar i klassen Player

4.1.1 Konstruktor:

- `Player(Sprite sprite)` - Skapar ett spelarobjekt på den position som `sprite` har, med den grafik som `sprite` använder. Använder konstruktorn i `Moving_Objekt`, och konstruktorn i `Moving_Object` använder kontruktorn i `Game_Object`.

4.1.2 Funktioner:

- (public) `bool update(Time time, Level level)` - Uppdaterar spelarobjektet baserat på spelaren knapptryckningar. Parametern `level` används för att lägga till projectiler i nivån när spelaren skjuter. Funktionen anropar `update` funktionen i `Moving_Object`.

4.1.3 Variabler:

- (private) `int health` - Antal liv spelaren har, [0, 3].

4.2 Arv från klassen Moving_Object

4.2.1 Funktioner:

- (public) `bool update(Time time, Level level)` - Uppdaterar objektets position. Returvärdet anger om objektet vill bli borttaget eller ej.

4.2.2 Variabler:

- (protected) `Vector2f velocity` - Anger hur snabbt, och i vilken riktning, ett objekt ska flytta på sig.

4.3 Arv från klassen Game_Object

4.3.1 Funktioner:

- (public) `void draw(RenderWindow window)` - Ritar ut ett objekt på skärmen.

4.3.2 Variabler:

- (protected) `Sprite sprite` - Innehåller objektets position och grafik.

5 Detaljbeskrivning av klassen Level

Klassen `Level` representerar en nivå i spelet som kan spelas. Ett `Level` objekt innehåller pekare till alla `Game_Objekt` och `Moving_Objekt` som nivån består av, och tillhandahåller två funktioner som objekten i en nivå anropar. `add_object` funktionen låter ett skjutande objekt lägga till projektilen i nivån, och `check_collisions` funktionen kollar om ett objekt kolliderar med något annat objekt i nivån.

5.1 Medlemmar i klassen Level

5.1.1 Konstruktor:

- `Level(String file_name, bool multiplayer)` - Anropar `load_level` med `file_name` som argument för att läsa in en nivå från en .csv fil. `multiplayer` parametern anger om nivån ska spelas av två spelare eller ej.

5.1.2 Funktioner:

- (public) `bool update(Time time)` - Anropar `update` på alla objekt i `moving_objects`, med både `time` och sig själv´som argument.
- (protected) `vector<Game_Object*> check_collisions(Game_Object object)` - Retunerar en vektor med pekare till alla objekt i nivån som kolliderar med `object`.
- (protected) `void add_object(Moving_Object* object)` Lägger till objektet `object` i `moving_objects` vektorn. Används av `Enemy` och `Player` klasserna för att lägga till projektiler i nivå.
- (private) `void load_level(String file_name)` - Läser in en nivå från en .csv fil.

5.1.3 Variabler:

- (private) `vector<Game_Object*> stationary_objects` - Vektor med alla stillastående objekt som nivån består av.
- (private) `vector<Moving_Object*> moving_objects` - Vektor med alla rörliga objekt som nivån består av. De två första objekten i vektorn ska vara `Player objekt. Det gör att \verb`Level| kan avgöra vilka objekt som är av typen `Player` utan att skapa ett direkt beroende mellan klasserna. Det ena `Player` objektet ska endast ritas ut och uppdateras om flerspelarläget är på.
- (private) `bool multiplayer` - Avgör om spelare två ska ritas ut och uppdateras.

6 Externa Filformat

Nivåerna läses in i spelet från .csv filer där varje nivå utgör en .csv fil. I .csv filen finns data gällande hur många tiles som nivåns bredd respektive höjd utgör. Filen innehåller också information om alla sprites koordinater för dess startposition.