LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

David Dumminsek, davdu153@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2022
Version 1.0

2022-11-25

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Skrev diskussion, UML och externfilformat. Första utkast slutfört och inlämnat	2022-11-25

2 Klasser

Vilka klasser som kommer användas i spelet.

2.1 Player

Denna klass ska modellera spelaren. Det ska vara möjligt för spelaren att röra sig, kollidera och skjuta projektiler.

Klassrelationer

1. **Projectile:** Player skapar Projectile objekt när spelaren skjuter.
2. **Game:** Vart Player objektet skapas.

Konstruktörer

1. **Player()** Sätter **life** (spelarens liv) till 2, **speed** (spelarens rörelsehastighet) till 0.1 och koordinaterna till det som är mitt i botten. Samt så konstrueras ett Projectile objekt.
2. **Player(int x, int y, float speed, int life, int damage):**

Variabler

1. **int coolDown**: Används för att sätta en timer mellan varje gång spelaren skjuter
2. **int coolDownMax**: Max värdet på timern.
3. Samt alla datamedlemmar från **Entity** basklassen

Funktioner

1. **void move():** Tar emot användarinmatning och uppdaterar koordinaterna med **speed** variablen. Om koordinaterna skulle hamna utanför spelet så ändras inte koordinaterna (kollision med fiende kommer hanteras i **Game** klassen).
2. **Projectile shoot():** Skapar **Projectile** objekt som startar med **Player** (spelarens) koordinater och rör sig rakt fram. Sedan returneras **Projectile** objektet.

2.2 Entity

Basklassen för alla klasser med koordinater, hastighet och liv.

Klassrelationer

1. **Projectile:** Subklass till Entity
2. **Player:** Subklass till Entity
3. **Enemy:** Subklass till Entity

Konstruktörer

1. **Entity(int x, int y, float speed, int life):** Tilldelar rätt värden till datamedlemmar. Denna konstruktor används också som en delegerande konstruktor för subklasserna.

Variabler

1. **int x:** X koordinater
2. **int y:** Y koordinater
3. **float speed:** Hur många koordinater objektet rör sig per rörelse.
4. **int life:** Hur mycket skada ett objekt kan ta innan den bör tas bort.

Funktioner

1. **void virtual move() = 0**: En pure virtual funtkion som kommer defineras på olika sätt i alla subklasserna
2. **bool takeDmg(int dmg)**: En funktion som tar bort int dmg parametern från int life datamedlemmen om **life** blir mindre än 0 så returneras true att objektet har 0 liv kvar.

3 Diskussion

Vår design är väldigt kompakt då Game klassen har väldigt mycket ansvar. Klassen tar hand om kollision, rendering och uppdatering av objekt. Det leder till att Game klassen har väldigt låg cohesion som är något man inte bör sträva efter. Med high cohesion blir koden mycket mer förstår bar samt blir det mycket lättare att isolera och hitta buggar. En åtgärd som skulle kunna göras är att dela upp **Game** klassen till mindre klasser. Till exempel kan en **ObjectManager** klass skapas som tar hand om alla objekt. Potentiellt skulle några ansvarigheter och datamedlemmar tilldelas till de andra existerande klasserna som t.ex **playerSprite** skulle kunna vara en datamedlem i **Player** klassen.

Fördelen med vår design är att den är väldigt simpel då det bara är 4 olika klasser så filuppdelningen blir inte så komplicerad och man kommer kunna komma igång snabbt och se hur det fungerar. Så det är bara att fokusera på funktionerna i **Game** klassen.

Rörelse skulle också kunnat förbättras med hjälp av SFML inbyggda scripting språk. Med vår nuvarande design så kommer **move** funktionen bli väldigt stor med alla if satset men det skulle kunna minskas med hjälp av scripts. Mycket mer komplicerade rörelse mönster skulle också kunnas göras då vår desing kan max skapa en projektil per tick.

4 Externfilformat

Tar användning av en json fil för att bestämma när fiender skapas, hur de rör sig och hur deras projektiler rör sig. Denna fil används alltså för att bygga upp en bana med att konstruera fiender vid specifika tider och positioner. Själva jsonfilen är uppbyggd av en json array med olika json objekt. I dessa json objekt finns det 11 olika variabler, dessa är:

1. **textureFile**: Bild filen för fienden
2. **tickSpawn**: Vid vilken tick fienden ska skapas (måste vara sorterad, där minsta är högst upp)
3. **x**: x koordinat där fienden ska skapas
4. **y**: y koordinat där fienden ska skapas

5. **speed**: Hur många y koordinater fiender rör sig på en tick
6. **life**: Hur mycket liv fienden har
7. **dmg**: Hur mycket skada ett fienderna projektiler ska göra.
8. **move**: En string som beskriver rörelsen
9. **projectile**: En string som beskriver rörelsen
10. **projectileSpeed**: Hur snabbt projektilen rör sig.
11. **fireRate**: Hur många ticks mellan varje projektil som skjuts

4.1 Rörelse

Rörelsa skapas med att hårdkoda olika rörelse mönster som sedan väljs med hjälp av de olika strängarna som beskriver rörelsen. Det åstakomms med enkla if else satser. Så om en ny rörelse ska skapas är det bara att navigera till **move** funktionen och lägga till en:
if else(movement == ny rörelse")
Sedan lägga den hård kodade koden under, ett litet tips ta hjälp av **movementTimer** variablen.