LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

David Dumminsek, davdu153@student.liu.se
Haris Basic, harba466@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2022
Version 0.1

2022-11-24

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	första utkast	2022-11-24

2 Klasser

Vilka klassar 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 använder ett projectile objekt för att beskriva hur Player projektilen fungerar. Om player delar koordinater med en projektil så dör spelaren.
2. **Enemy:** Om ett Player objekt delar koordinater med ett Enemy objekt dör spelaren.
3. **Game:** Vart Player objektet skapas.

Konstruktorer

1. **Player()** Sätter **life** till 2, **speed** till 0.1 och koordinaterna till det som är mitt i botten. Samt så konstrueras ett Projectile objekt.
2. **Player(Projectile a):** Är lik default constructorn utom att en valfri Projecile objekt kan användas.
3. **Player(const& Enemy) = default** Copy constructor
4. **Player(const&& Enemy) = default** Move constructor

Variabler

1. **int x:** x koordinater för spelaren
2. **int y:** y koordinater för spelaren
3. **float speed:** hur många x eller y koordinater spelaren rör sig per tick.
4. **int life:** hur många liv spelaren har.
5. **Projectile pro:** Projectile object som beskriver hur projektilen rör sig och hur mycket skada den gör:'

Funktioner

1. **Enemy& operator=(const& Enemy rhs) = default:** Copy constructor
2. **Enemy& operator=(Enemy&& rhs) = default:** Move constructor
3. **Enemy():** Destructor
4. **void move():** Tar emot användarinmatning från game loopet och updaterar koordinaterna med **speed** variablen. Om koordinaterna skulle hamna utanför spelet så ändras inte koordinaterna (fiende kollidering kommer hanteras i **Game** klassen).

5. **void shoot():** Skapar **Projectile** objekt som startar med **Player** koordinaterna och rör sig rakt fram i en hastighet beroende på objektet.
6. **void die():** Ändra koordinaterna till ursprungs koordinaterna och ändra **life** med -1.

2.2 Game

Denna klass tar hand om funktionerna som används i gameloopet: Klassen ska huvudsakligen rendera och updatera.

Klassrelationer

1. **Projectile:** Håller koll på alla projektilers koordinater.
2. **Player:** Håller koll på spelar koordinaterna.
3. **Enemy:** Håller koll på alla enemy koordinater.

Konstruktorer

1. **Game():** Läser alla level filer och lägger det i **level** vektorn. Skapar **Player** objektet.

Variabler

1. **vector<map> level:** En vektor av **map** typen som innehåller bl.a när fiender skapas och värden för att konstrua ett **Enemy** objekt.
2. **int tick:** En integer som updateras efter varje gameloop och används för att bestämma när fiender skapas, var de befinner sig beroende på deras rörelse funktion. Och när spelaren har nått slutet av banan.
3. **vector<Enemy> enemies:** En vektor med alla fiende objekt.
4. **vector<Projectile> enemyProjectile** En vektor med alla projektiler från fiender.
5. **vector<Projectile> playerProjectile** En vektor med alla projektiler från spelaren.

Funktioner

1. **void collisionCheck():** Jämför alla fiender och fiende projektil koordinater med spelarens. Om koordinaterna är nära nog varanndra så kallas **Player.die()** funktionen. Jämför alla spelar projektiler med alla fiender koordinater och om projektilerna är nära nog en fiende så kallas **Enemy.takeDmg()** funktionen.
2. **void update():** Ändrar alla koordinater på spelaren och fiender med deras **move()** funktioner. Kallar också **spawnEnemy()**, **updateProjectile**, **collisionCheck** och **victory**.
3. **void render():** Renderar spelaren och alla fiender och projektiler.
4. **void victory():** Kollar om **tic** är över en viss gränns och alla fiender är förstörda. Om det stämmer så startas nästa bana.
5. **void spawnEnemy():** Kollar nuvarande **map** variabel i **level** vektorn om **tick** är lika med **tickSpawn**. Om det är det så skapas ett **Enemy** objekt med hjälp av alla **map** värden.
6. **void updateProjectile():** Går igenom varje **Enemy** objekt och kallar deras **shoot()** funktion för att skapa **Projectile** objekt. Lägger till nya skapade projektiler i **enemyProjectile** variablen. Uppdaterar koordinaterna på alla projektiler med hjälp av **Projectile.move()** funktionen.

3 Diskussion

3.1 Fördelar

3.2 Nackdelar

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 konstruea fiender vid specifika tider och positioner. Själva jsonfilen är uppbygd av en json array med olika json object. I dessa json object finns det 7 olika variabler, dessa är:

1. **id**: Ett unikt nummer för varje fiende
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. **spd**: Hur många y koordinater fiender rör sig på en tick
6. **movement**: Ett json objekt som beskriver en polynomfunktion, används för rörelsemönster för fienden
7. **projectile**: Ett json objekt som beskriver en polynomfunktion, används för rörelsemönstret för projektilerna fienden skjuter

4.1 Rörelse

En polynom av den fjärde graden används för att simulerara rörelsen för fiender och projektiler. Som ett json objekt beskrivs funktionen som följande:

```
"projectile": {"a":0, "b":0 ,"c": 0, "d":1, "f":0}
```

Figur 1: Json objekt som beskriver en polynom funktion

Alla variabler i Json objectet är då konstanterna i polynomfunktionen. Just i figur 1 så beskriver json objektet en projektil som åker snedd längst skärmen. Det json objektet beskriver alltså funktionen:

$$y = x$$

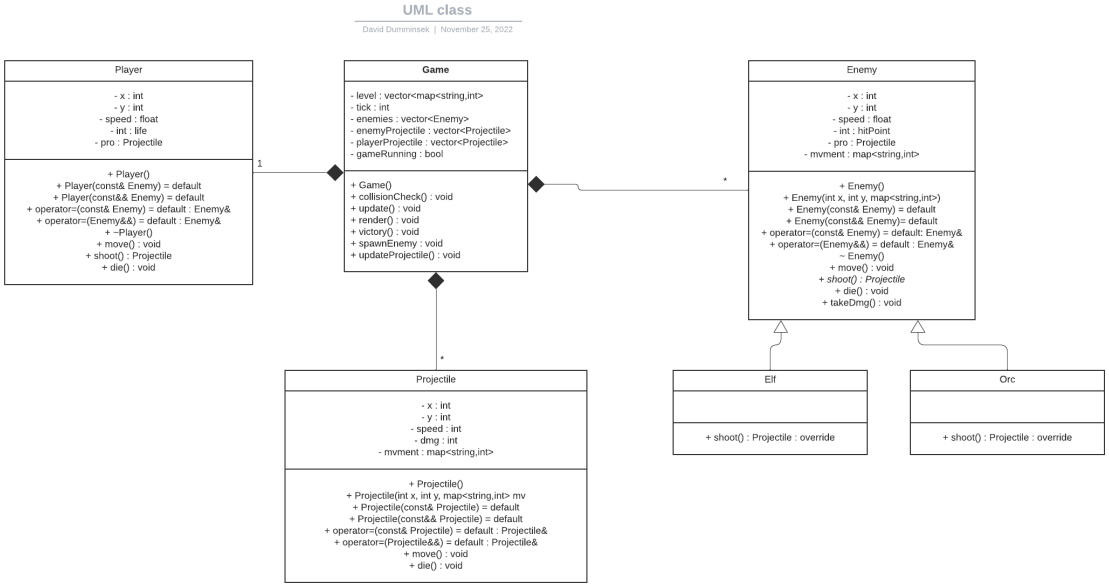
Figur 2: Enkel linjär funktion

$$ax^4 + bx^3 + cx^2 + dx + f$$

Figur 3: Polynom av fjärde graden

5 UML Klass Diagram

Klassdiagramet för spelet:



Figur 4: