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

### Konstruktorer

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(Projectile a):** Är lik default constructorn utom att ett valfri Projectile 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 och/eller y koordinater spelaren rör sig per tick.
4. **int life:** hur många liv spelaren har.
5. **Projectile pro:** Projectile objekt 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 uppdaterar koordinaterna med **speed** variablen. Om koordinaterna skulle hamna utanför spelet så ändras inte koordinaterna (kollidering med fiende kommer hanteras i **Game** klassen).

5. **void shoot():** Skapar **Projectile** objekt som startar med **Player** (spelarens) koordinater och rör sig rakt fram i en hastighet beroende på objektet.
6. **void die():** Ändrar koordinaterna till de ursprungliga koordinaterna och minskar **life** med 1 (life - 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 koordinaterna för alla projektiler
2. **Player:** Håller koll på spelarens koordinater.
3. **Enemy:** Håller koll på alla fientliga enheters koordinater.

**Konstruktorer**

1. **Game():** Läser alla level filer (spelets banor) 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 konstruera 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 koordinater för fiender samt deras projektiler med spelarens koordinater. Om koordinaterna är tillräckligt nära varandra så kallas **Player.die()** funktionen. Jämför även koordinaterna för spelarens projektiler med fiendernas koordinater. Om spelarens projektiler är tillräckligt nära en fiende så kallas **Enemy.takeDmg()** funktionen.
2. **void update():** Ändrar koordinater för 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 gräns och om alla fiender har eliminerats. 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

De främsta fördelarna med vår design är att det är lätt att bygga upp en grund för spelet rent kodmässigt samt att det är lätt att läsa av klasser på ett överskådligt sätt. Resultatet av detta är att det går att se hur allting fungerar på ett enkelt och översiktligt sätt. En positiv effekt av vår design är att filuppdelning blir lättare, vilket i sin tur gör att även kompilering blir betydligt lättare.

Det finns dock ett flertal nackdelar med vår design. Om någonting skulle gå fel vid utvecklandet av spelet så blir det svårt att fastställa exakt var en ändring behöver göras eftersom varje klass innehåller många olika delar. Ännu en nackdel är att om vi vill ändra på en funktion som är lik en funktion i en annan klass så måste vi göra den ändringen i alla liknande klasser (eftersom att vi ej använder polymorfi).

Om vi exempelvis skulle göra en ändring i en move-funktion så behöver vi även göra ändringar hos andra move-funktioner då de är uppbyggda på ett liknande sätt. Ytterligare problem med vår design är att det är svårt att hålla koll på alla olika funktioner (eftersom att det finns väldigt många olika) och om vi i framtiden skulle behöva dela upp klasser i flera klasser så kommer det bli en väldigt rörig och tidskrävande process.

Det finns väldigt mycket upprepning i vår kod eftersom att vi inte använder oss av polymorfi. Ett sätt vi skulle kunna ha gjort saker och ting annorlunda är att vi exempelvis skulle kunna ha använt polymorfi till allt som rör på sig.

Ett exempel på vad man skulle kunna ha gjort är att göra så att det fanns en överklass till de som de ärver datamedlemmar av funktioner ifrån. Detta hade gjort det betydligt enklare att fastställa var ett potentiellt fel har skett samt göra det enklare att göra ändringar i koden utan att behöva göra ändringar i alla liknande klasser. Användning av polymorfi hade minskat mängden upprepning i vår kod, vilket skulle göra den ännu mer läsbar och enklare att tyda och förstå. Det hade blivit enklare att hålla koll på alla olika funktioner.

# 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 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}
```

Figure 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:

$$y = x$$

Figure 2: Enkel linjär funktion

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

Figure 3: Polynom av fjärde graden

# 5 UML Klass Diagram

Klassdiagrammet för spelet:

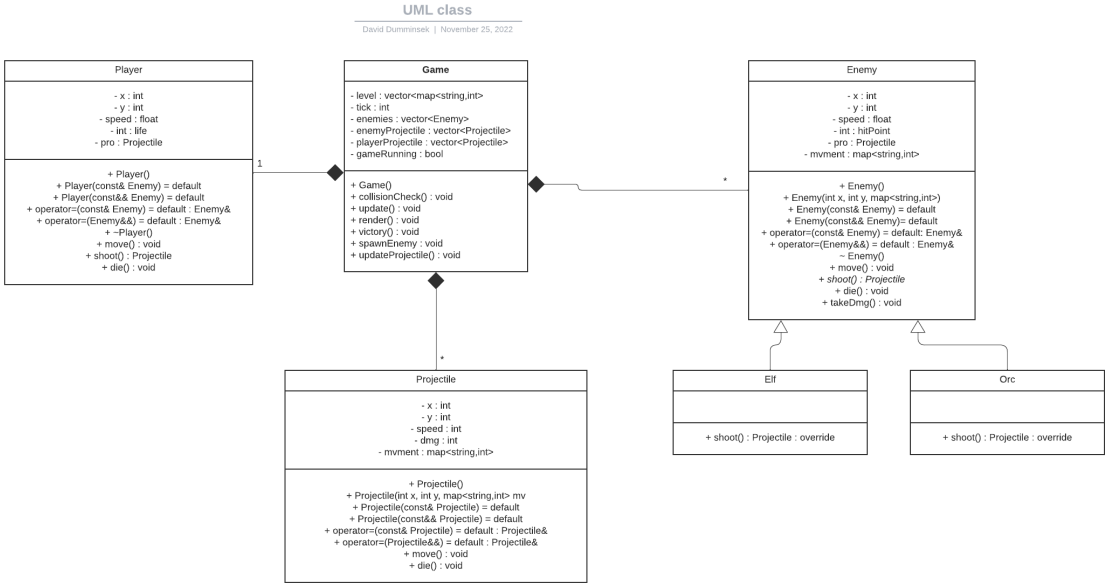


Figure 4: