Projektbeskrivning

BoxHead: Ghetto edition

2021-03-08

Projektmedlemmar:
Theodor Larsson <thela038@student.liu.se>
Filip Johnsson <filjo653@student.liu.se>

Handledare:
Joel Kågemyr <joeka960@liu.se>

Innehåll

Projektplan

Läs först:

- https://www.ida.liu.se/~TDDD78/labs/2021/project/intro
- https://www.ida.liu.se/~TDDD78/labs/2021/project/select
- https://www.ida.liu.se/~TDDD78/labs/2021/project/documents

Ni väljer själva vilken sorts projekt ni vill utföra, men det finns vissa begränsningar som man behöver tänka på. Läs om detta på ovanstående sidor. Det är viktigt för att ni inte ska måla in er i ett hörn med ett projekt som inte är lämpligt för kursen.

Projektplanen skriver ni i samband med första inlämningen, gärna under tiden ni arbetar på sista labben för att göra det möjligt att få kommentarer innan projektstart. Små kompletteringar kan göras senare, men försök få med så mycket som möjligt redan från början.

1. Introduktion till projektet

Vi tänker utveckla ett zombie shooter spel där man ska slåss mot vågor av zombies, plocka upp pengar, ammunition med mer för att överleva. Man börjar med en simpel pistol, sedan kan man köpa bättre och bättre vapen som till exempel en UZI eller en AK-47: a. Det finns två varianter av zombies, dem som går fram och slår dig till döds och zombie poliser med pistoler som skjuter på dig. Det finns inget sätt att ”vinna” spelet, utan det handlar om att överleva mot den konstant ökande mängden zombies.

2. Ytterligare bakgrundsinformation

Koncept och implementation är relativt simpelt. Finns inte någon ytterligare information som krävs utöver att första java.

3. Milstolpar

#	Beskrivning
1	`Skelett för spellogik (gameloop, initialisering av spelet osv)`
2	`Interface sprite (Rotation, position, textur)`
3	`Entity abstrakt klass implements sprite`
4	`Entity hanterare (skapar, uppdaterar och ritar entities)`
5	`Rita upp bakgrund`
6	`Entity subklasser, fiender, spelare, väggar, skott`
7	`Kollisonshantering`
8	`Skapa spelarklassen`
9	`Fiende subklasser`

10	Fiende generator/spawner
11	Skjuta
12	Fiender dör när slut på liv
13	Kunna förlora spelet
14	Spara score
15	Implementera fler vapen
16	Kunna köpa nya vapen
17	Bilar som ligger på marken som hinder
18	hotseat??
19	breakable walls???
20	
21	
22	
23	
…	

4. Övriga implementationsförberedelser

```
Boxhead

    Gameloop

    Initialisering

Interface Sprite

    Rotation

    Position

    Textur

    Storlek

    Draw()

Abstract class Entity implements Sprite

    Update()

    Kollision

Abstract class LivingEntity implements Entity

    Health

    Hastighet
```

```
    Vapen
Class Bullet extends Entity
    Hastighet
    Skada
Class Wall extends Entity
Abstract class Collectable extends Entity
Class CCash extends Collectable
    $$$ till spelare
Class CAmmo extends Collectable
    Ammo till vapen
Class Enemy extends LivingEntity
    Target
    AI
Class Player extends LivingEntity
    Score
    $$$
(((Class BreakableWall extends LivingEntity))) kanske implementera



interface class Weapon (indata: ägarobjekt)
    Ägare
    Hastighet
    Skada
    Reload time
    Ammo
    Textur
    Skjuta()
Abstract class enemyWeapon implements projectileWeapon
    Weapon range
Abstract Class playerWeapon implements projectileWeapon
    Pris
Class minigun extends playerWeapon
```

```
Class EntityHandler
    Entity list
    clearEntities list
    clearEntities()
    updateEntities()
    drawEntites()
    addEntity()
    removeEntity()
```

5. Utveckling och samarbete

Vi har samarbetat förr och har en bra metod.

Projektrapport

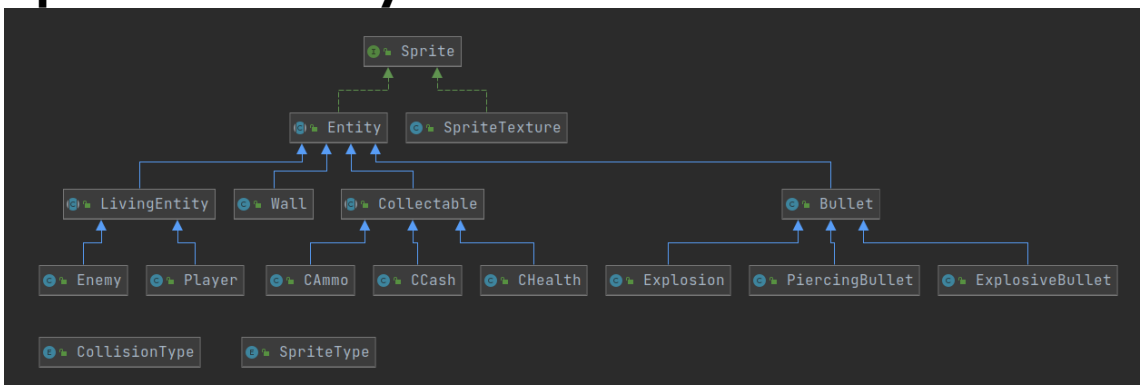
6. Implementationsbeskrivning

6.1. Milstolpar

Vi hann med alla nedskrivna milstolpar förutom hotseat och breakable walls. Många funktioner finns inte med i milstolpar eftersom dem kom vi på under projektets gång. Vi lyckades inte implementera rotation till den utsträckningen vi velat. T.ex får bilarna väldigt konstig hitbox om man försöker ändra på deras rotation, trots att själva bilden roterar. Det var helt enkelt inte värt besväret att lösa detta problem eftersom det inte alls var centralt till projektet. Vi hann inte fixa en pausfunktion. MeleeWeapons används bara för fienderna, hade gått att implementera ett eller flera vapen till spelaren.

6.2. Dokumentation för programstruktur, med UML-diagram

Sprite och Entity överblick



Allting som syns på skärmen i spelet implementerar interfacet Sprite. Med den implementerad har objektet alltid ett sätt att rita ut sig själv med Sprite.draw() funktionen. Sättet detta sker på är att det finns ett SpriteHandler objekt där man kallar på SpriteHandler.add() funktionen med ett objekt som implementerar Sprite som parameter. För detta SpriteHandler objekt har även en funktion SpriteHandler.getIterator() och med den får man en iterator över alla Sprite objekt som är tillagda i SpriteHandler. Sedan kan då GameComponent, som sköter Swing arbetet, iterera över alla Sprites och kalla på deras Sprite.draw() funktion och på så sätt rita ut alla objekt samtidigt, men ändå ha att hur objekten ritas ut på ett väldigt decentraliserat sätt.

Eftersom Sprite endast är ett interface går det inte att skapa ett objekt av det, men om man bara vill ha en bild, en text eller en linje utritad utan att den ska kunna interagera med andra objekt, då används SpriteTexture-klassen. Den är en simpel klass som implementerar Sprite utan någon extra funktionalitet.

Alla fysiska objekt i spelet tillhör abstrakta klassen Entity. Den implementerar Sprite,

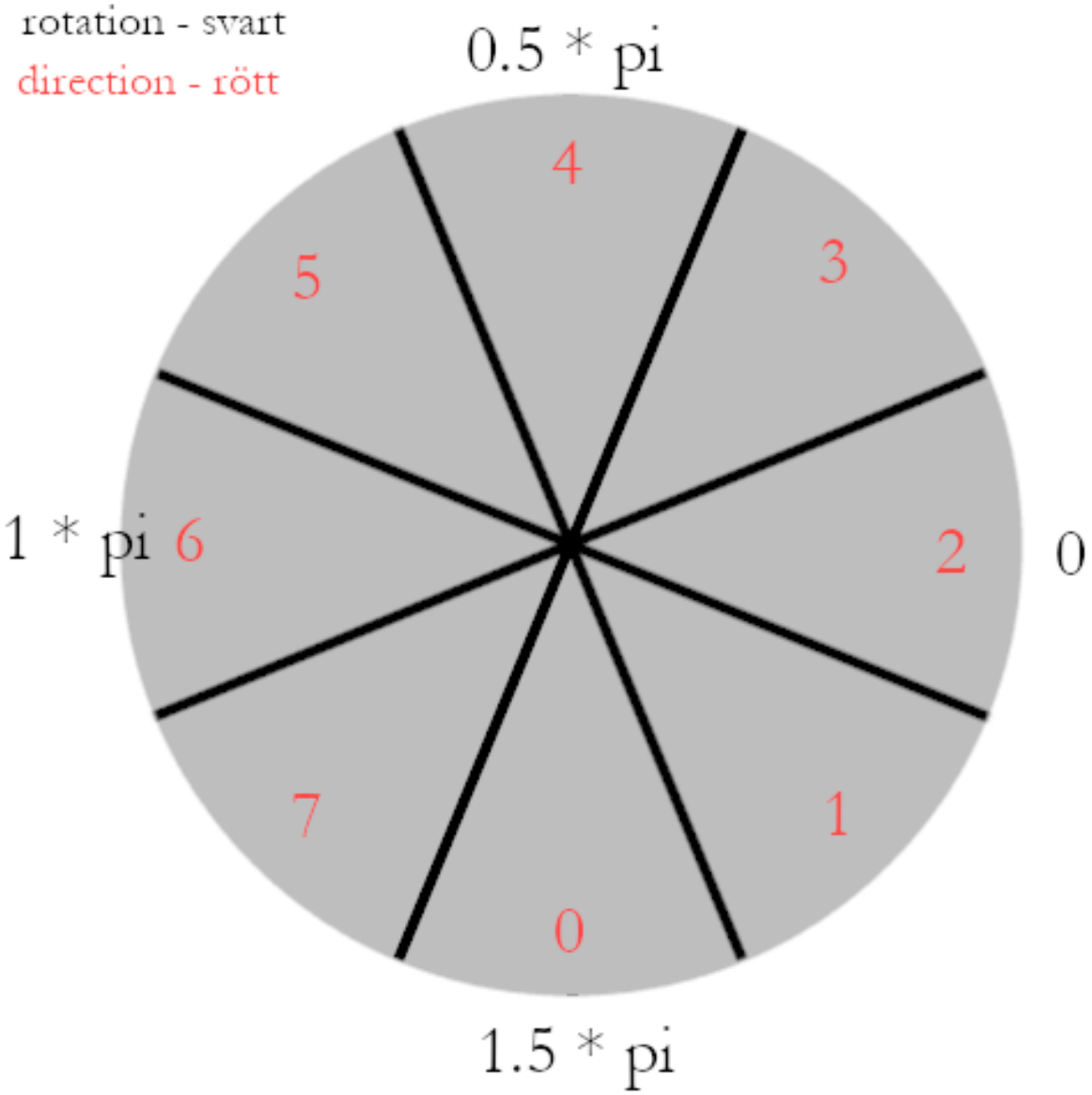
men utöver det låter den objektet interagera med andra objekt. Den gör detta genom att alla Entities har ett statiskt COLLISION_TYPE-fält beroende på vilken subklass det är, till exempel har LivingEntity typen CollisionType.LIVING_ENTITY. När en kollision mellan två objekt detekteras (kommer till hur detta händer snart) anropas bådas Entity.onCollide() funktion. I denna finns en switch-sats med alla CollisionTypes och en default på en Entity.baseCollide.

Entity.OnCollide() funktionen har som basimplementation att skicka upp kollisionen till sin superklass med vilken CollisionType det var. Till exempel, basimplementationen i Player-klassen är att den kallar på LivingEntitys collide funktion, som kallar på Entitys baseCollide funktion. Detta betyder att man kan lägga kollisionslogik i mitten av hierarkin där det passar bäst istället för att behöva upprepa sig i lägsta subklasserna.

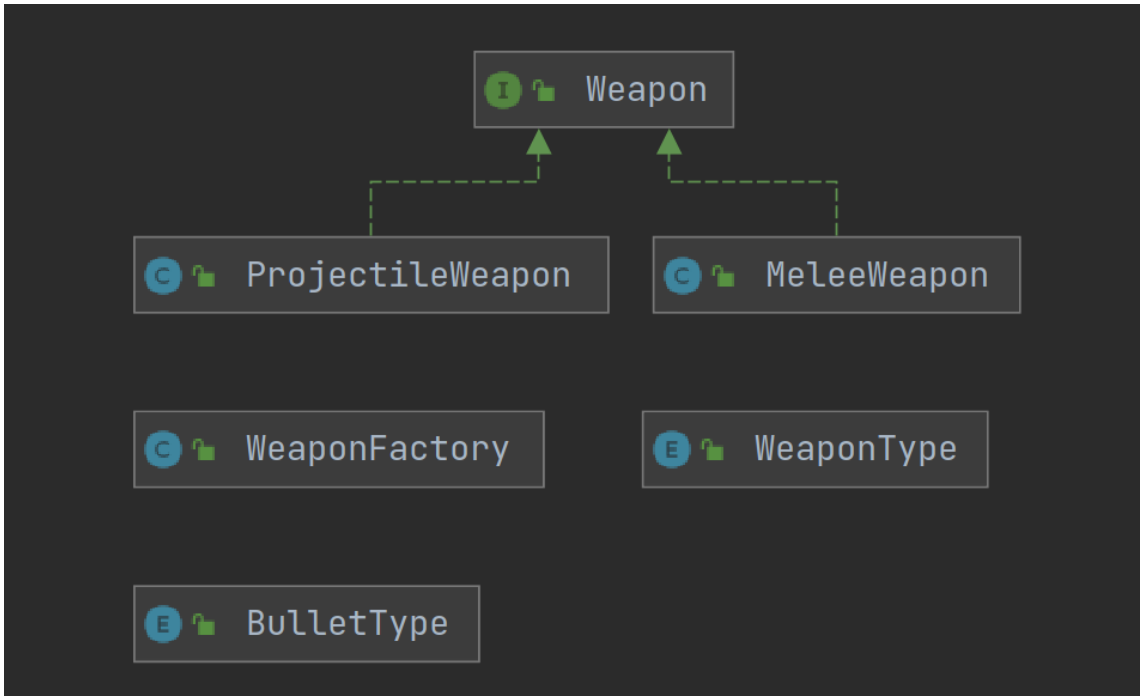
Entity har även en annan viktig funktion, Entity.update(). Den uppdaterar allt som man vill ska uppdateras i objektet. Den utökas i subklasserna för att sköta all logik och att uppdatera allt som de lägger till.

Eftersom Entity och Sprites är rätt annorlunda i hur de implementeras och används finns det även en EntityHandler-klass. Den har även en EntityHandler.add() funktion och EntityHandler.getIterator() funktion. Men den har en EntityHandler.update() vilket itererar över alla Entities som ligger i EntityHandler-objektet (vilket borde vara alla Entities) och kallar på deras Entity.update()-funktion samt att den tittar vilka Entities som kolliderar med varandra. Den gör detta genom att anropa alla Entities Entity.getCollisionArea()-funktion och sedan titta efter en intersection. Detta låter individuella Entities bestämma hur stor och vilken form deras ”hitbox” för kollision ska vara med override.

Entity har ett fält Entity.rotation som är ett värde mellan 0 – 2 pi, men LivingEntity har ett annat snarlikt fält LivingEntity.direction. Förhållandet mellan dessa fält går åt ett håll, LivingEntity.rotation bestämmer LivingEntity.direction, men direction har ett värde på 0 – 7. Detta är på grund av att det finns 8 vinklar av karaktären ritade i sprite sheeten. LivingEntity.direction är alltså indexet som används för att få ut rätt textur på karaktären. Kombinera detta med att noll värdet är på olika delar av cirkeln och att direction har en ytterligare 1/16 * pi offset från rotation för att få mer intuitiva riktningar på karaktärerna blir detta lätt förvirrande. Se diagram nedan för illustration av skillnaden.



Weapon överblick



En annan central del av spelet är vapen. För detta finns det ett interface Weapon som ställer några grundläggande krav på vad som behövs av ett vapen. Ett vapen ska implementera update(), draw(), getCorrectAttackOffset(), getWeaponType() och onWeaponAttack().

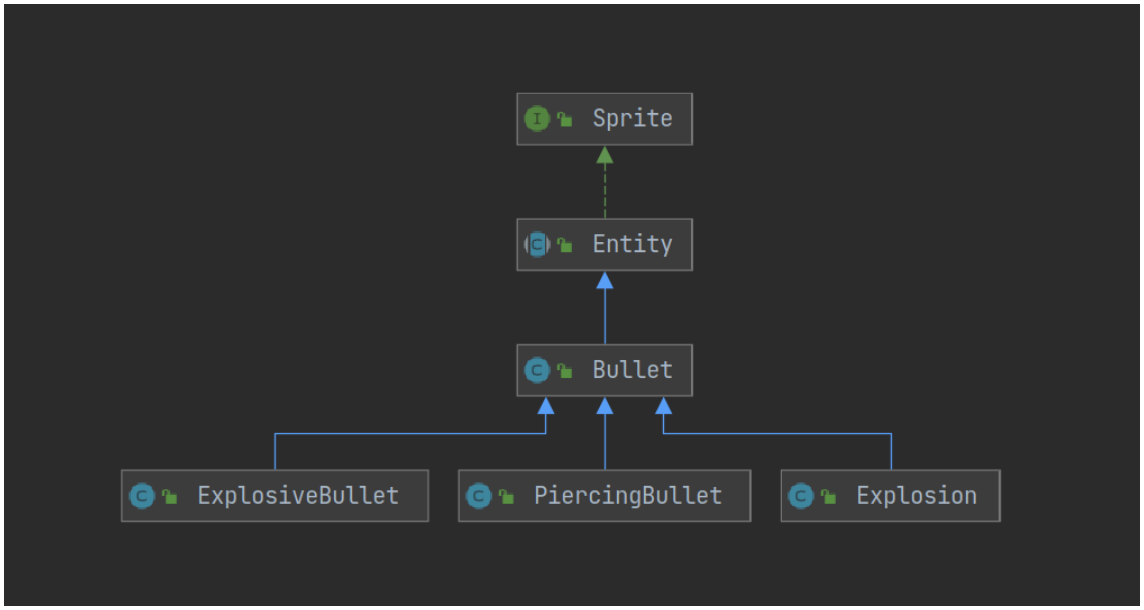
getCorrectAttackOffset()

I Weapon.getCorrectAttackOffset() hämtas vilken offset som ska användas om vapnet skjuter projektiler. Offseten hämtas från listan ProjectileWeapon.bulletOffset, medan MeleeWeapon.getCorrectAttackOffset returnerar en offset på 0.

(Att MeleeWeapon ens har denna funktion håller dörren öppen för att i framtiden kunna implementera ett melee vapen som också skjuter projektiler.)

Betydelsen av ProjectileWeapon.bulletOffset är att när vapnet skapar en projektil ska den inte skapas från det övre vänstra hörnet på karaktären, utan det ska ju komma ut ur pipan på vapnet. Offseten är hur långt ifrån pipan är från vänstra övre hörnet beroende på vilken LivingEntity.direction karaktären har.

onWeaponAttack()



I Weapon.onWeaponAttack() skapas en attack. Beroende på om vapnet är MeleeWeapon eller ProjectileWeapon kommer dessa attacker se väldigt annorlunda ut.

Ett MeleeWeapon skapar en Area runt sig som sedan jämförs i EntityHandler.entitiesInArea() för att hitta om någon Entity befann sig i det området. MeleeWeapon tittar sedan om ägaren är en Player eller inte och beroende på det gör den skada på Enemies eller Player i Arean. MeleeWeapon är dock endast implementerat för Enemies för stunden.

Ett ProjectileWeapon däremot går igenom en switch-sats och skjuter sedan ut en Entity.Bullet, Entity.PiercingBullet eller Entity.ExplosiveBullet beroende på ProjectileWeapon.bulletType som har ett värde av enumen BulletType. ProjectileWeapon ger dessa Bullets sina initiala värden och sedan har den ingen kontroll över dem längre, utan de läggs in och uppdateras i EntityHandler.

Input överblick

All input från tangentbordet hanteras med hjälp av paketet input. Den mest centrala klassen är KeyHandler, som har en samling med objekt som implementerar GameKeyListener. KeyHandlern lyssnar efter input från tangentbordet, och uppdaterar alla GameKeyListeners varje gång användaren ger en input. Det som skickas med till varje GameKeyListener vid varje uppdatering, är ett objekt av typen KeyEvent, samt en Map som innehåller den nuvarande statusen för varje knapp.

Ett objekt som implementerar GameKeyListener behöver den publika funktionen onKeyEvent. Det är den här funktionen som körs av KeyHandler varje gång användaren ger en input. KeyEvent, som skickas med som parameter i onKeyEvent, innehåller getters för Key och KeyState. Med hjälp av dessa så vet man vilken knapp som har ändrat läge, samt om knappen blivit nedtryckt eller uppsläppt. En andra parameter i onKeyEvent är en Map som innehåller det nuvarande läget för alla knappar.

Enumen Key innehåller alla typer av tangenter som används i spelet, till exempel SHOOT och RELOAD. Key är alltså inte en specifik knapp på tangentbordet, utan istället en viss typ av input. På det sättet så kan man ändra vilken tangentbordstangent som representerar en viss Key, utan att ändra i något av objekten som implementerar GameKeyListener. Det går alltså att ändra så att RELOAD aktiveras av ‘H’ istället för ‘R’, eller i framtiden implementera funktion för att använda en spelkontroll som aktiverar RELOAD.

För att lägga till en GameKeyListener i KeyHandler så används funktionen KeyHandler.addKeyListener. När en GameKeyListener laggts till i KeyHandler så kommer den få uppdateringar för varje Key som KeyHandler lyssnar efter.

I det här spelet så är det bara Player som behöver reagera på input från tangentbordet, och det är därmed bara Player som implementerar GameKeyListener.

7. Användarmanual

Spelar karaktären styrs med W-A-S-D och använder mellanslag för att skjuta.

Riktningen på vart spelaren skjuter styrs med hjälp av muspekaren.

För att köpa nästa vapenuppgradering, tryck på E-knappen.

För att växla mellan de vapen du redan äger, tryck på Q-knappen.

För att ladda om det vapnet du håller i, tryck på R-knappen.

Strax efter att spelet startas kommer första vågen av zombies, de kommer från utanför kanten på skärmen. De jagar alltid dig och du måste skjuta och döda dem för att överleva. När en zombie dör har den en chans att släppa ett av tre användbara föremål.

Money stack – Ger spelaren pengar som används för att köpa nya vapen.

Healthkit – Ger tillbaka health till spelaren.

Ammo box – Ger 3 magasin av ammunition till det vapnet spelaren håller i.

Nere i vänstra hörnet ser man hur mycket pengar och score spelaren har samlat på sig.

Vapenkostnader:

Uzi – 50

AK-47 – 150

RPG-7 - 250

Nere i det högra ser man hur många skott vapnet har kvar i magasinet och hur många magasin spelaren har i reserv.

I det vänstra övre hörnet ser man vilken våg man är på. Det kommer en ny våg antingen efter en viss tid har passerat eller om alla zombies är döda.

Det finns två typer av fiender i spelet. Walkers som går mot spelaren och gör skada om de kommer nära. Sen finns det poliser som skjuter på håll mot spelaren och tar mer stryk innan de dör. Med smart manövrering kan man blockera deras skott med andra zombies däremot!

Det finns en del bilar som står stilla på gatan. Dessa blockerar både zombies och spelarens rörelser. Zombies är inte särskilt intelligenta och fastnar gärna bakom dessa, använd detta till din fördel.