LINKÖPINGS UNIVERSITET

TDP003 Projekt: Egna datormiljön

Dokumentmall

Författare

Nils Bark, nilba048@student.liu.se
Hadi Ansari, hadan326@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2020
Version 0.1

2020-11-27

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Första utkast skapat	2020-11-27

2 Beskrivning av Entity

Nedan kommer en lista över alla funktioner och variabler som finns i klassen `Entity`.

2.1 Variabler

- `sf::Vector2f location`: Innehåller två koordinater av typen float som representerar spelarens position på spelplanen.
- `sf::Texture texture`: Tar in en bild och använder den som textur för ett objekt.
- `sf::Sprite sprite`: Fungerar som figurens kropp och kan täckas av en textur.
- `float speed`: Används för att bestämma objekts hastighet.
- `int health`: Bestämmer objektets livspoäng.
- `int width och int height`: Bestämmer objektets storlek.

2.2 Funktioner

- `virtual void Tick(sf::Time const & delta) = 0`: Uppdaterar objektens position. Den implementeras av de härledda klasserna och beteer sig där på olika sätt. Till exempel för Big plane så subtraheras x-koordinaten med ett värde som är bundet till hastigheten och tiden som har gått sedan förra uppdateringen.
- `virtual bool Want_shoot() = 0`: Returnerar true om objektet vill skjuta, annars false. Till exempel returneras alltid false för bomber eftersom de aldrig skjuter, medan true returneras när spelaren trycker ned mellanslag (om andra faktorer tillåter).
- `virtual Entity* shoot() = 0`: Returnerar en entity-pekare som pekar på `player_bullet` eller `enemy_bullet` beroende på om det är spelaren eller en fiende som ska skjuta. Har ingen definition i klasser som aldrig kommer att använda shoot().
- `bool kill_me()`: Kontrollerar om ett objekt ska förstöras (till exempel om det befinner sig utanför spelplanen) och returnerar true om så är fallet.
- `virtual void collision(vector<entity*> objects) = 0`: Tar en lista som inehåller entity-pekare till alla andra spelobjekt, utöver det som kallat på funktionen, och kontrollerar vilket av dem objektet har kolliderat med. Har flera olika effekter beroende på vilket objekt som har kolliderat med ett annat. Till exempel förlorar spelaren liv om den kolliderar med en fiende.
- `virtual string get_type() = 0`: Returnerar objektets typ i form av en sträng.

`Entity` är en `abstract struct` som alla understående klasser i spelet ärver ifrån. Detta innebär att alla dess variabler och funktioner är tillgängliga direkt i de härledda klasserna. `Entity` låter oss därmed uppdatera alla andra objekt i spelet. Detta sker via funktionen `tick()` som tar in en parameter av typen `sf::Time` och uppdaterar positionen för alla objekt på skärmen. `Tick()` är en `virtual` funktion som kan överskrivas i de härledda klasserna. Tack vare tidsberäkningen som sker innuti funktionen är vi oberoende av spelets bildfrekvens och tiden det tar att hantera alla händelser i loopen. Alla datamedlemmar förutom `location` i

basklassen kommer att initieras av de härledda klassernas constructor. till exempel kommer spelarens `speed` att initieras när spelaren skapas och inte innan.

3 Beskrivning av Player

Nedan kommer en lista över alla funktioner och variabler som finns i klassen `Player`.

3.1 Variabler

- `bool sht`: Är true då spelaren trycker på mellanslag (och därmed vill skjuta), false annars.
- `float shoot_speed`: Bestämmer tiden som måste passera efter att spelaren har skjutit innan den får skjuta igen.
- `sf::Clock clock1`: Är en klocka som mäter tiden som har gått mellan spelarens skott.
- `sf::Time t1`: Sparar tiden som har passerat enligt clock1 i form av sekunder.
- `bool shield`: Är true om spelaren har shield-effekten aktiv.
- `sf::Clock shield_clock`: Räknar tiden från och med att spelaren plockade upp en shield-powerup

3.2 Funktioner

- `sf::Vector2f process_event(sf::Time delta)`: Hanterar händelser åt spelaren. Till exempel vilka tangenter som trycks ner och vad som då ska hända.
- `string get_shield_time()`: Returnerar den kvarstående tiden från `shield_clock`.
- `bool has_shield()`: Returnerar true om spelaren har en shield aktiv, annars false.

`Player` representerar det objekt som användaren kommer att ha kontroll över under spelets gång. Det är en härledd klass av `Entity` och ärver därmed alla datamedlemmar och funktioner därifrån. Den har också en känna-till-relation till `player_bullet` och `Power-ups` härledda klasser. Player använder sig av process_event för att uppdatera `sht` (ifall det tillåts av de andra faktorerna) eller uppdatera spelarens `location`. Player överskriver `tick()` för att kunna läsa in en ny `location` som `process_event()` har uppdaterat och sätta spelarens position baserat på det. Förutom de variabler den ärver har `Player` även en `int health` som avgör spelarens liv. I spelarens konstruktor initieras spelarens `health, location, speed, shoot_speed, sht, t1, shield, sprite` och `texture`. `kill_me()` kollar här om spelarens liv hamnar på eller under 0. Om detta händer förstörs spelaren och spelet är över.

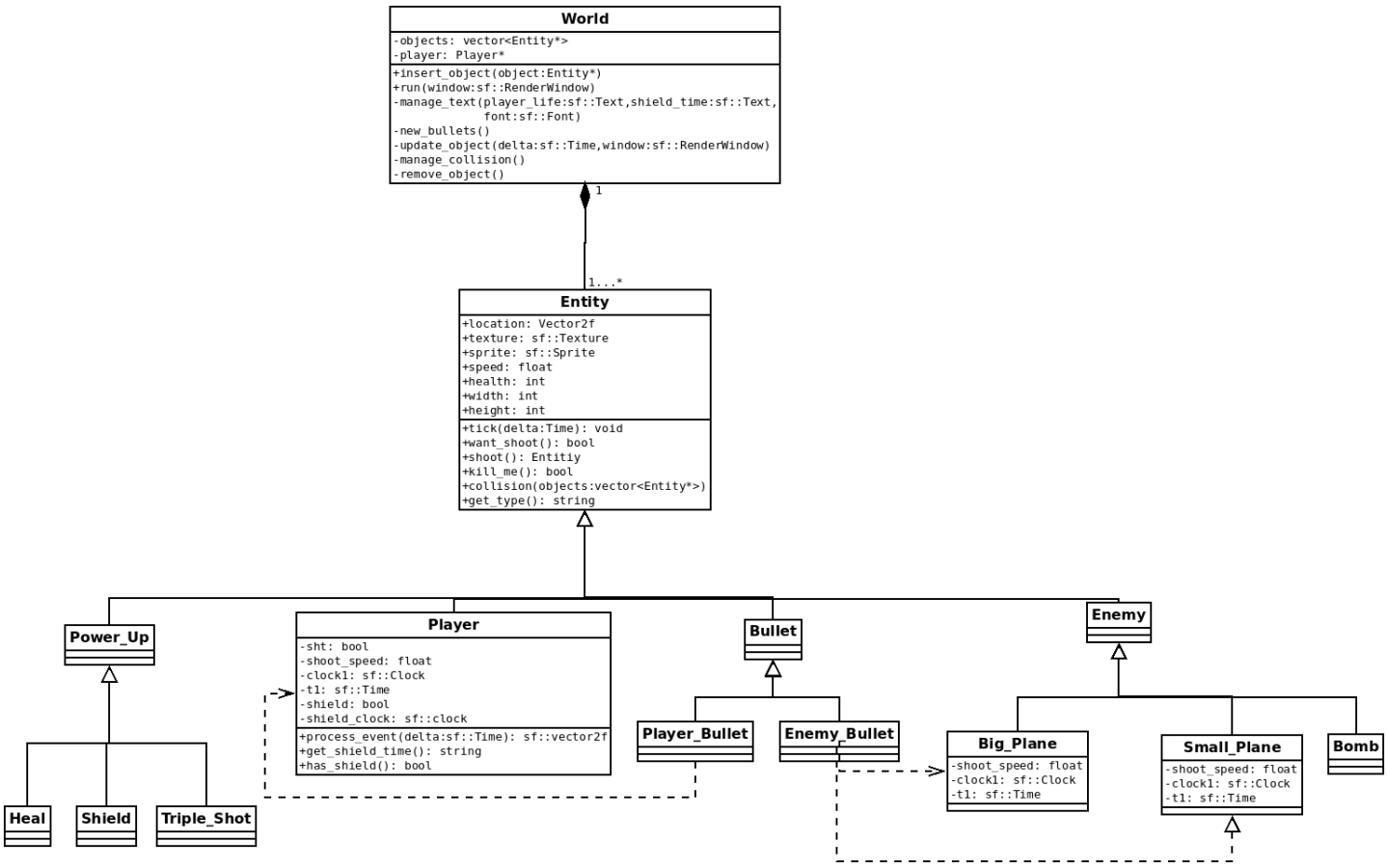
Spelarens kollision med andra objekt kontrollerar ständigt under spelets gång via `collision()` från `entity`. Baserat på vilket objekt som spelaren kolliderar med kan flera olika saker hända. Till exempel tar spelaren skada vid kollision med fiender (förutom då spelaren har en shield aktiv), medan den diverse aktiverar effekter vid kollision med power-ups. Den enda kollisionen som ignoreras av spelaren är med dess egna skott (och fienders skott ifall spelaren har en shield aktiv).

4 Diskussion

Så pass tidigt i projektet finns inte mycket nyanserad diskussion att ha kring för- och nackdelar av det som har gjorts hittills. Dock är en fördel att alla objekt i spelet fördelas olika klasser som har korrekt relationer till varandra, vilket leder till att spelet hänger ihop på ett bra sätt.

5 Externa filformat

Planen är att banor ska läsas in från externa filer, så som JSON eller txt. Om bör-kravet att implementera topplistor över poäng uppnås så är planen att spara poängen i en JSON fil.



Figur 1: UML-diagram över spelet