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 Game_Object

`Game_Object` är en `abstract struct` som alla klasser i spelet ärver ifrån. `Game_Object` låter oss därmed uppdatera alla andra object i spelet. Detta sker via funktionen `tick()` som tar in en parameter av typen `sf::Time` och uppdaterar med hjälp av den positionen för alla objekt på skärmen. `Tick()` är en `virtual` funktion som kan överskridas i de härledda klasserna. Tack vare tidsberäkningen är vi oberoende av spelets bildfrekvens och tiden det tar att hantera alla händelser i loopen. Datamedlemmarna i basklassen kommer att initieras av de härledda klassernas struktur, till exempel kommer spelarens `speed` att initieras när spelaren skapas och inte innan.

- `Center` är en variabel av typen `Vector2f` som sparar en figurs origo.
- `Speed` är en variabel av typen `float` som används för att bestämma objekts hastighet.
- `Sprite` är en variabel av typen `sf::Sprite`. Den fungerar som figurens kropp och kan täckas av en textur. `sprite` används även för att kontrollera kollision med andra objekt.
- `Texture` är en variabel som tar in en bild och använder den som textur för ett objekt.

3 Beskrivning av Player

`Player` representerar det objekt som användaren kommer att ha kontroll över under spelets gång. Det är en härledd klass av `Game_Object` och ärver därmed alla datamedlemmar och funktioner därifrån. Den har också en aggregations-relation till `Bullet` och `Power-ups` härledda klasser. Den har en funktion `process_event()` som läser in nedtryckta tangenter via `sf::Keyboard::Key` och använder sig av den informationen för att antingen kalla på `shoot()` eller uppdatera spelarens `location`. Den överskrider `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 liv (till 3), `location` och `speed`. Här tar även `texture` och importerar den bild som ska representera spelaren och täcker `sprite` med den.

För att skjuta mot de fiender spelaren kommer stöta på använder sig `Player` av funktionen `shoot()`. Den kallas på när `process_event` läser ett nedtryckt mellanslag och skapar då ett bullet-objekt som färdas rakt åt höger med konstant hastighet. Vid kollision med fiender ska det skada dem, annars förstör objektet då det hamnar utanför spelplanen.

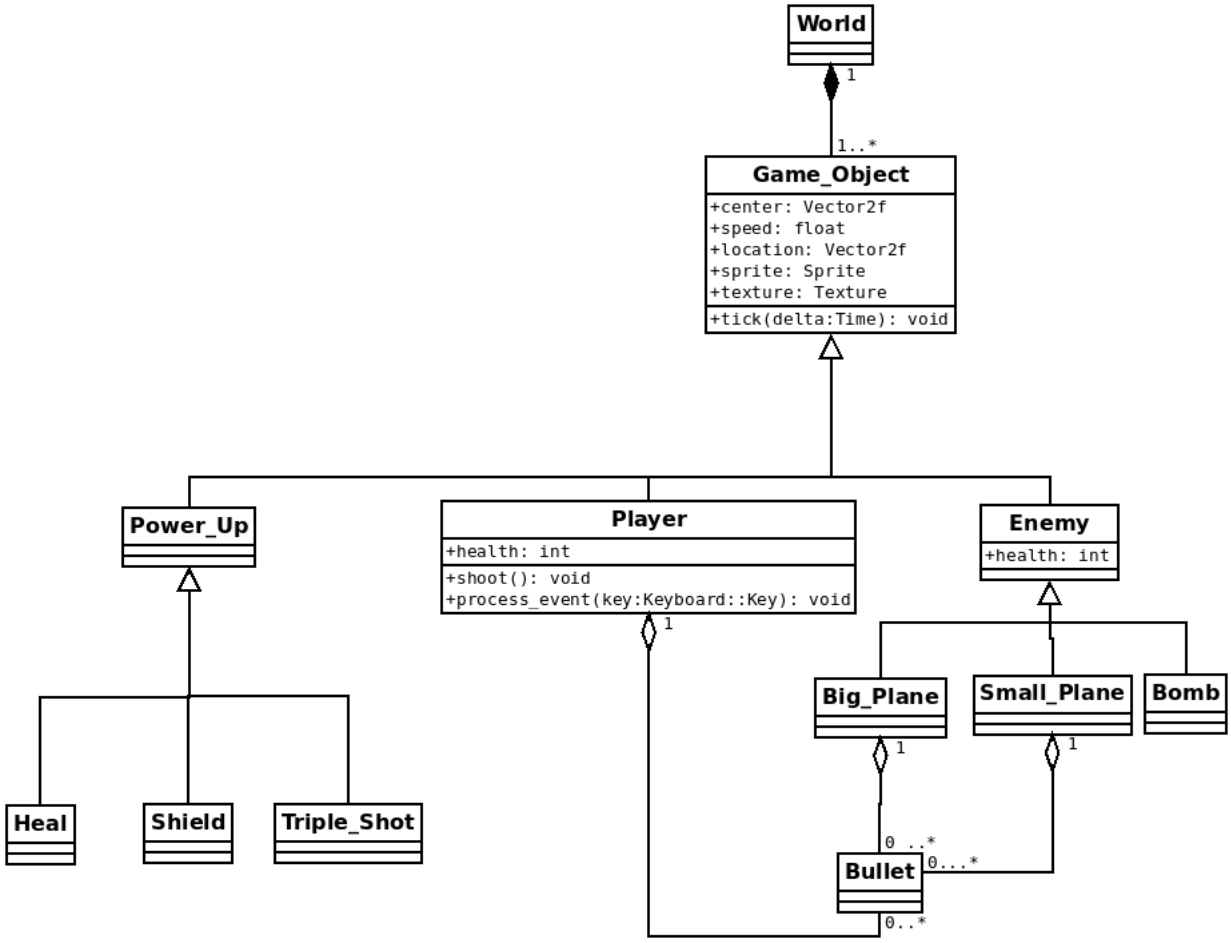
Det ska finnas en kollisionshanterande funktion i klassen som med hjälp av spelarens `location` kontrollerar om den har kolliderat med något annat object och utför då korrekt respons till kollisionen (eller ignorerar kollisionen ifall ingen respons krävs).

4 Diskussion

Så pass tidigt i projektet finns inte mycket nyanserad diskussion att ha kring för- och nackdelar av det lilla 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