SDL2 – Dragon flame hell

Mitt fokus låg i att utveckla AI, bakgrund, spawn- och game manager. Mestadels i AI eftersom det är ett av det områden jag vill utforska och lära mig mer om, till detta behövde vi en spawn manager som skapade och skicka ut fiender slumpmässigt i spelet. Utöver det så löste jag även vår scrollande bakgrund och game manager, game manager städade mestadels upp i main filen.

My part

Min del i projektet består av följande filer:

- Enemy.h & Enemy.cpp
- Background.h & Background.cpp
- SpawnManager.h & SpawnManager.cpp (Grimm och jag satt en del tillsammans)
- GameManager.h & GameManager.cpp

Sen utförde jag några mindre optimeringar, error-hantering och pill i följande filer:

- Engine.cpp (Error hantering i “Initialize-funktionen”, Quit åtgärder i “EngineDestroy - funktionen”)

Enemy

Innehåller allt som har med fienden att göra, rörelsemönster, skott, kollisionsbox, rendering och uppdatering av detta under spelets gång. Fienden är ett *“game object”* och ärver därför den klassens funktioner och variabler, alla *“game objects”* hanteras inuti *“game world”*-klassen. Där uppdateras, renderas, förstörs och spawn:as alla *“game objects”* under spelets gång.
Fiendeklassen består i stora drag av:

- Konstruktor som kallas på när en fiende skapas, här bestäms positionen, rörelsemönster, skott typ, filväg för hämtning av sprite. Den får även med en kollisionsbox och i vilken riktning fienden rör sig i.
- Uppdaterings funktion som uppdaterar fiendens kollisionsbox utifrån fiendes position samt aktiverar rätt rörelsemönster och skott typ. Även timern startas, som håller reda på hur ofta fienden får skjuta.
- Renderingen som renderar fienden utifrån position, önskad storlek och filväg. Själva renderingen sker via en funktion i *“Engine.cpp”*, där skapas själva ytan och texturen. Här hanteras även error rörande själva renderingen av fiendens texture.
- Funktioner som hanterar rörelsemönster och skott typ.

Background

I likhet med fiendeklassen innehåller bakgrundsklassen allt som har med bakgrunden att göra, position, storlek, rendering och uppdatering av detta under spelets gång. Bakgrunden är också ett *“game object”* och ärver därför den klassens funktioner och variabler, alla *“game objects”* hanteras inuti *“game world”*-klassen. Där uppdateras, renderas, förstörs och spawn:as alla *“game objects”* under spelets gång.

Bakgrundsklassen har även två "SDL_Rect"-variabler som används för att skapa den sömlösa scrollande bakgrundsbilden, samt några definierade värden för måtten på bakgrundsbilden och skärmen. Bakgrundsklassen består i stora drag av:

- Konstruktor som kallas på när en bakgrund ska skapas, här bestäms positioneringen, storleken och filvägen för bakgrundsbilden. Här sätts även måtten för det två rektanglarna som används för den scrollande bakgrunden.
- Uppdaterings funktionen som skapar illusionen av en sömlös scrollande bakgrundsbild, genom att ge den ena rektangeln en hastighet på y-axeln. För att sen nollställas när rektangeln nått halvvägs ned på bakgrundsbilden.
- Renderingen som renderar bakgrundsbilden utgår ifrån original rektangeln och.....? . Själva renderingen sker via en funktion i *"Engine.cpp"*, där skapas själva ytan och texturen. Här hanteras även error rörande själva renderingen av fiendens texture.

Spawn Manager

Den här klassen tar hand om att spawn:a fienden, en ganska simpel klass med ett syfte. Den har två funktioner kopplat till att spawn:a fiender, där vi egentligen bara använder den första eftersom den andra var version ett.
Funktionen spawn:ar en fiende utifrån ett visst tidsintervall baserat på delta time, får en slumpmässigt vald position i x-led, slumpmässigt rörelsemönster och skott typ. Vilket jämfört med version ett av funktionen är en betydlig optimering, där vi istället spawn:a tre "game objects" med ett bestämt antal gånger varje typ av fiende skulle spawn:a.
Anledningen till varför vi tog fram version två var för att få en "infinity-loop" av spawn:ande fiender, vilket behövdes för att skapa en bättre speldesign.

Game Manager

Vår "Game manager" är enkelt sagt en klass som hanterar allt som vi först hade i vår "*main*"-klass, alltså vår "*game loop*". Den består i huvudsak av tre funktioner. "*Start*", "*update*" och "*quit*", som sen kallas på i "main"-klassen. Det är här spelet körs och uppdateras under "*runtime*", det är även här delta time sätts.
Utöver det initialiserar vi vår "*game world*"-klass, "*spawn manager*"-klass och tid, skapar pointers till vår spelare, bakgrund och fienden.

Strength and weakness

En svaghet är att vi har olika typer på hur vi namnger variabler och bygger upp klasser m.m. vi kunde ha lagt tid på att lägga upp en struktur för namngivning för att ha en mer sammanhängande koduppbyggnad. Vissa namn för klasserna är också något förvirrande som t.ex. vi har en "game world" och en "game manager", vilket är lite otydligt om det gör samma sak eller vad skillnaden är.
Styrka med vårt program är att koden i klasserna är relativt enkla och raka att förstå, bra kommenterat i det klasser vi la till i den befintliga motorn.

Improve the system

Med mer tid kunde vi optimerat hur vi använder oss av delta time, just nu har vi två olika. En som sätts i engine och den andra som är en egen klass, vi kunde även ha jobbat mer konsekvent med namngivningen kring klasser och variabler.

Teamwork

Den främsta utmaningen för vår grupp var att vi alla var nya i både C++ och SDL2, vilket bidrog till att vi kanske inte kom lika långt som andra grupper. Även om vi inte hann utveckla allt som vi ville fick vi ändå utmanas och utvecklas på vår egen nivå och takt.
Utöver det så hade vi också en del utmaning med att använda *"Sourcetree"* samt att lägga upp en struktur över arbetet. Så här i efterhand borde vi skrivit ett gruppkontrakt med tydlig struktur för arbetssättet t.ex. tider, kontakt, gruppmöten, deadlines m.m. Det blev svårt att arbeta i grupp när kommunikationen var dålig, närvaron var upp och ner och otydliga mål/deadlines. Något jag får ta med till nästa projekt.
Vårt största misstag var att vi valde en motor som vi inte riktigt förstod till fullo när vi började, det hade troligtvis varit bättre om vi hade tagit fram en egen motor eller arbetat i en mer lättförståelig. Vilket också blev den största lärdomen från kursen/projektet, för om du är med och bygger upp motorn har du större koll på hur du kan använd den. Vilket leder mig in på höjdpunkten med kursen/projektet.
Höjdpunkten med kursen/projektet var att vi faktiskt fick jobba i och med en motor, vilket gav en större förståelse över hur en motor är konstruerad och fungerar. Istället för att arbeta i en färdig motor som Unreal eller Unity, där jag själv måste lära mig motorns funktioner och verktyg istället för att faktiskt vara med och konstruera motorn. För mig gav det i alla fall ökad förståelse och djupare kunskap kring både vad en motor är och koncepten bakom den. Det gav även mig ett ökat intresse för att i framtiden arbete med motorprogrammering.