Vores applikation

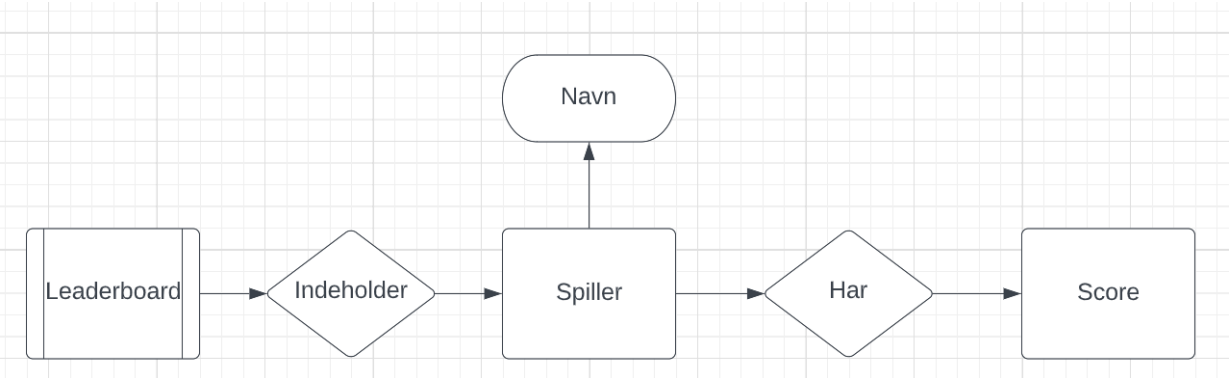
Vores idé var at lave et scoreboard/leaderboard til et spil ved hjælp af firebase, så alle spillere kunne se hinandens score og konkurrere mod hinanden om den højeste score. Vi har taget udgangspunkt i et spil, som vi udviklede med Unity til vores arkademaskine i teknologi. Spillet gemmer scoreboardet lokalt på computerens harddisk ved brug af Unitys Player Prefs, som gør det muligt at gemme ting mellem spil sessioner. Scoreboardet virker godt når det kun er en computer som spiller spillet, men det er skidt hvis andre personer skal se andres highscores da det hele ligger lokalt på en computer.

Vores produkt skal altså bygge videre på det spil, som vi allerede har lavet og koble en form for cloud database sammen med det enkelte spil, som gør at spillet har et online scoreboard, i stedet for at hvert spil har deres egen lokale scoreboard. Det online scoreboard vil gøre at hvert spil vil kunne have et synkroniseret scoreboard, så alle spillere med en internetforbindelse kan se hinandens score. Vi vil forsøge at anvende KISS på produktet, som gør det så simpelt som muligt at bruge, så brugere ikke behøver at sætte sig ind i nogle tekniske detaljer, som kan være svære for en normal bruger, men derimod nyde at det bare virker.

Designtænkning Proces

Som cloud database vil vi anvende firebase, da det er en meget simpel NoSQL cloud database, og da det er en af kravspecifikationerne for dette projekt. Firebase gemmer data i en JSON format, som passer godt til vores formål med at gemme scoreboard data. Ved brug af Firebase kan man se et opdateret scoreboard lige meget hvilken computer man spiller spillet på.

Når vi skal skrive data til databasen skal vi først have en skabelon for hvordan vores datastruktur skal være. Vi ved at der er et navn associeret med en unik score, som den person med det navn fik. Vi kan opbygge et E/R diagram som beskriver vores datastruktur bedre. E/R diagrammet vil hjælpe os når vi skal til at implementere tilføjelsen til spil koden, ved at gøre det klart hvilke nogle data vi skal have styr på, og eventuelt hvordan vi skal håndtere dataen i koden. Spillerdataen består af et initial/navn, samt en score den person fik, det kan forklares med et E/R diagram:



[1]: E/R diagram af hvilke nogle datapunkter som er vigtige (Lavet i Lucidchart).

På diagrammet kan man se hvilke nogle informationer leaderboardet består af. Leaderboardet indeholder flere spillere som hver har et navn og en score associeret med navnet.

E/R diagrammet vil altså hjælpe os i koden til hvordan vi skal strukturere dataen inden vi sender det afsted til databasen. Ud fra diagrammet, kan vi generere et simpelt JSON objekt, eller struct i C#, som er en datatype vi kan bruge. Ved at aflæse diagrammet kan vi se at hver indgang i scoreboardet har et navn og score. En datatype som indeholder alle de vigtige informationer kan laves således:

```
10  [Serializable]
11  public struct scCol {
12          public int score;
13          public string scoreName;
14          public string initial;
15
16          public scCol(int _score, string _scoreName, string _initial) {
17              this.score = _score;
18              this.scoreName = _scoreName;
19              this.initial = _initial;
20          }
21  };
```

Koden viser et struct, som indeholder alle de informationer som beskrevet i E/R diagrammet [1], samt en simpelt constructor til at generere et nyt objekt af denne datatype.

KISS eller Keep it simple, stupid kan blive brugt på vores projekt, da det ville være smart at gøre det nemt for forbrugeren at bruge vores produkt. Ved at gøre det nemt for forbrugeren, ville det give en bedre oplevelse, men også gøre det nemmere at læse og skrive koden. Den første tanke vi fik var en helt almindelig arkadespil score, som kunne gemme score i en database. Vi tænkte det var simpelt både for forbrugeren at bruge, og kodning bag det.

Implementering

Når spilleren dør eller klarer spillet, så kommer man tilbage til hovedmenuen hvor han eller hun har et virtuelt keyboard som spilleren kan bruge til at skrive sine initialer på maksimalt tre bogstaver. Efter brugeren har trykket enter med sine initialer, sender klienten (spillet) en PUT request til vores firebase server med

spillerens initialer og score i requesten. Spillerdataen der bliver sendt følger den datastruktur, som der blev beskrevet tidligere [1] ved at lave et objekt af “scCol”, som indeholder initialerne og scoren. Vi har skrevet en asynkron metode, som tager spiller dataen og en callback funktion, som bliver kaldt når vi får svar tilbage fra serveren:

```
427     public static void Postdata(scCol data, string scID, PostdataCallback callback) {
428         RestClient.Put<scCol>($"{databaseURL}scores/{scID}.json", data).Then(response => {
429             callback();
430         });
431     }
```

[2]: PostData() metoden bliver kaldt når vi skal uploade noget til firebase.

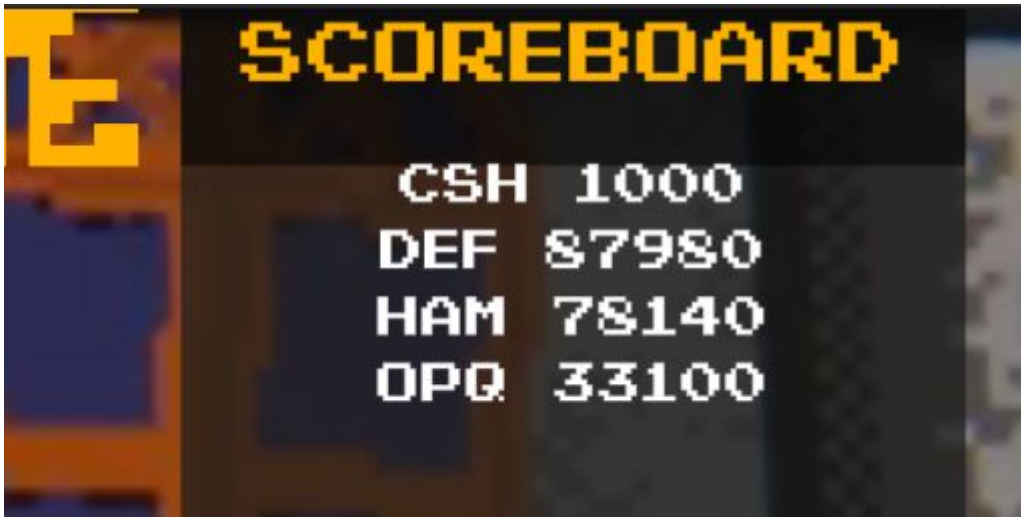
Spillerdataen vil dermed blive sendt til Firebase og gemt på deres server som vi låner. Derefter bliver en callback funktion kaldet, så vi ved hvornår dataen er blevet modtaget hos serveren.

Efter vi har sendt dataene og fået svar fra serven, kan vi begynder at opdatere det visuelle scoreboard i spillet ved at sende en GET request til Firebase og bruge et library som hedder full serializer til at konvertere den string vi får fra serveren til et Dictionary objekt. Koden kalder callback funktionen med resultatet og vi opdaterer dermed text elementet i Unity med resultatet:

```
439     public static void Getscores(GetscoresCallback callback) {
440         RestClient.Get($"{databaseURL}scores.json").Then(response => {
441             var responseJson = response.Text;
442             var data = fsJsonParser.Parse(responseJson);
443             object deserialized = null;
444             serializer.TryDeserialize(data, typeof(Dictionary<string, scCol>), ref deserialized);
445             var scores = deserialized as Dictionary<string, scCol>;
446             callback(scores);
447         });
448     }
```

[3] : GetScores() metoden bliver kaldt når vi skal have downloade scoreboardet fra firebase.

Da vi designede scoreboardet fulgte vi KISS ved at gøre det rigtigt simpelt. Det er nemt for spilleren fordi han eller hun slet ikke skal gøre noget for at få det til at virke. I spillet er det ikke nødvendigt at have nogle “fancy” knapper, som i virkeligheden ikke rigtig har noget formål. Simpliciteten af scoreboardet giver generelt spilleren en bedre oplevelse, da der ikke er en masse ekstra ting de skal tage højde for. På billedet forneden kan man se scoreboardet med initialer på personen som fik scoren og hvilken score de fik.



Test
Vi har lavet en video, som der viser vores produkt blive testet. Man kan se videoen på YouTube med linket forneden:
https://www.youtube.com/watch?v=VHMSWNITLD0

I testen demonstrerer vi at spillet virker. Vi starter i hovedmenuen af spillet hvor man kan se det nuværende scoreboard, som spillet får fra firebase. Vi begynder derefter et spil hvor vi indsamler point ved at dræbe monstre, samle pickups op osv. Når vores spilkarakter dør, bliver vi smidt tilbage til hovedmenuen, hvor vi bliver præsenteret med et keyboard, hvor vi kan skrive vores initial. Efter vi skriver “INF”, sender vi spiller dataen til firebase (se [2]). Derefter forsvinder keyboardet og vi genindlæser scoreboardet (se [3]), som viser den nye score på scoreboardet.

Efter at det lykkes i spillet åbner vi firebase konsollen hvor vi kan se at der er blevet tilføjet en score som hedder “INF”, som indeholder den rigtige fra spillet. Til sidst for at være sikker genstarter vi spillet, hvor vi observerer at ændringen bliver vist på scoreboardet.

Konklusion

Vi havde nogle problemer med at få firebase til at virke ordentligt. Det skyldes at dokumentationen for hvordan man integrerede firebase med unity var ringe og outdated. Det tog derfor længere tid end det oprindeligt skulle have taget. Efter at have løst problemerne med Firebase, virkede det fint og var pålideligt. I sidste ende fik vi altså Firebase til at gøre det vi ville have den til, altså at modtage data fra spillets scoreboard, som vi igen kunne hente fra den, og displaye det til spilleren.