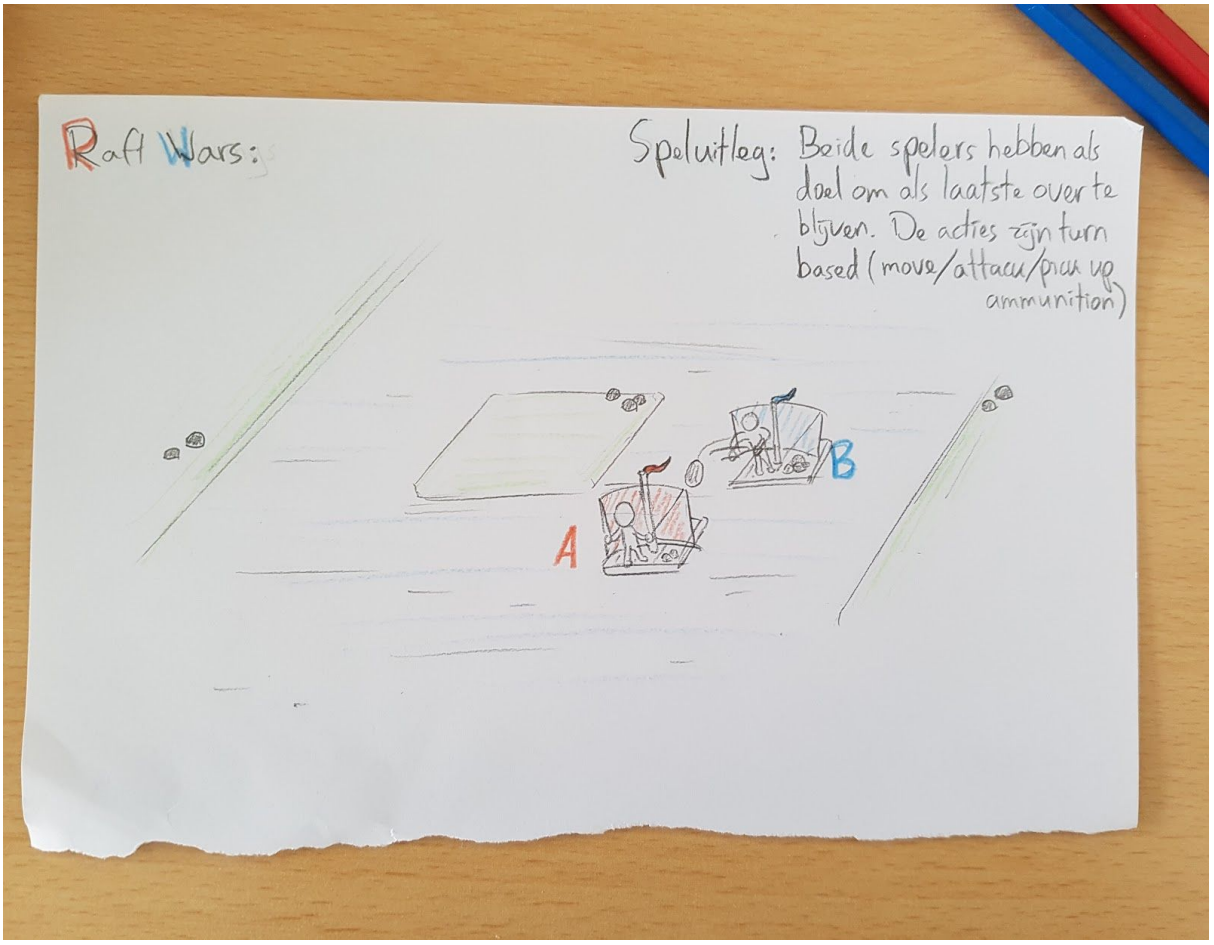
Multiplayer

Inleiding

Voor deze periode hadden wij de opdracht om een turn based multiplayer game te maken. Ik heb hier in de eerste les het volgende ontwerp voor bedacht:



Het idee hierbij was dat beide spelers een vlot hebben. Om de beurt kunnen ze daarmee een actie uitvoeren. Doel van het spel is dat je als laatste vlot blijft drijven. Dit kun je bereiken door andere vlotten tot zinken te brengen.
De acties die de spelers kunnen uitvoeren zijn ‘varen’, ‘schieten’ en ‘munitie oppakken’. Ik heb later de actie ‘munitie oppakken’ geschrapt vanwege een tijdsgebrek. Verder heb ik wel volledig dit ontwerp aangehouden.

Ik heb ervoor gekozen om met UNET te werken, omdat ik hier al eerder een keer mee gewerkt had en omdat het naar mijn mening best goed gedocumenteerd is.

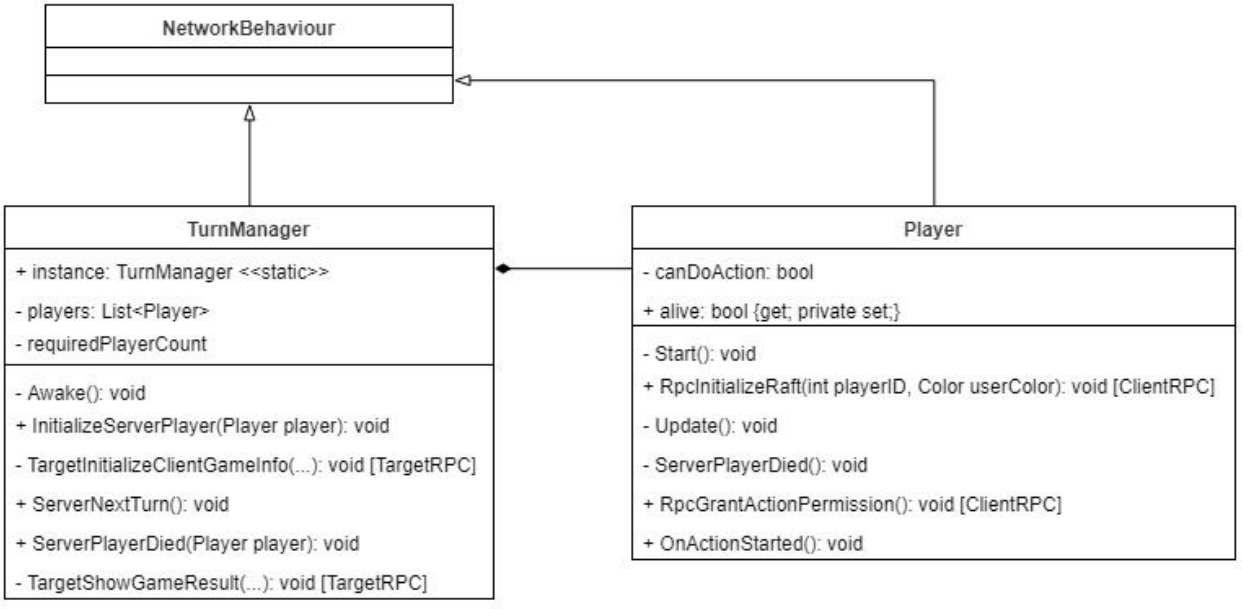
Technical Challenges

Bij het maken van dit spel kwam ik op een aantal uitdagingen. De grootste hiervan was de TurnManager. Om dit werkend te krijgen heeft mij behoorlijk veel tijd en moeite gekost, maar toen dit eenmaal werkend was, ging de rest ook wel relatief soepel.
In de kern bestaat de TurnManager uit een script dat aan de server-kant alle spelers bijhoudt. Deze spelers subscriben zichzelf aan de TurnManager als ze gespawned zijn. Dat gebeurd met de volgende functie:

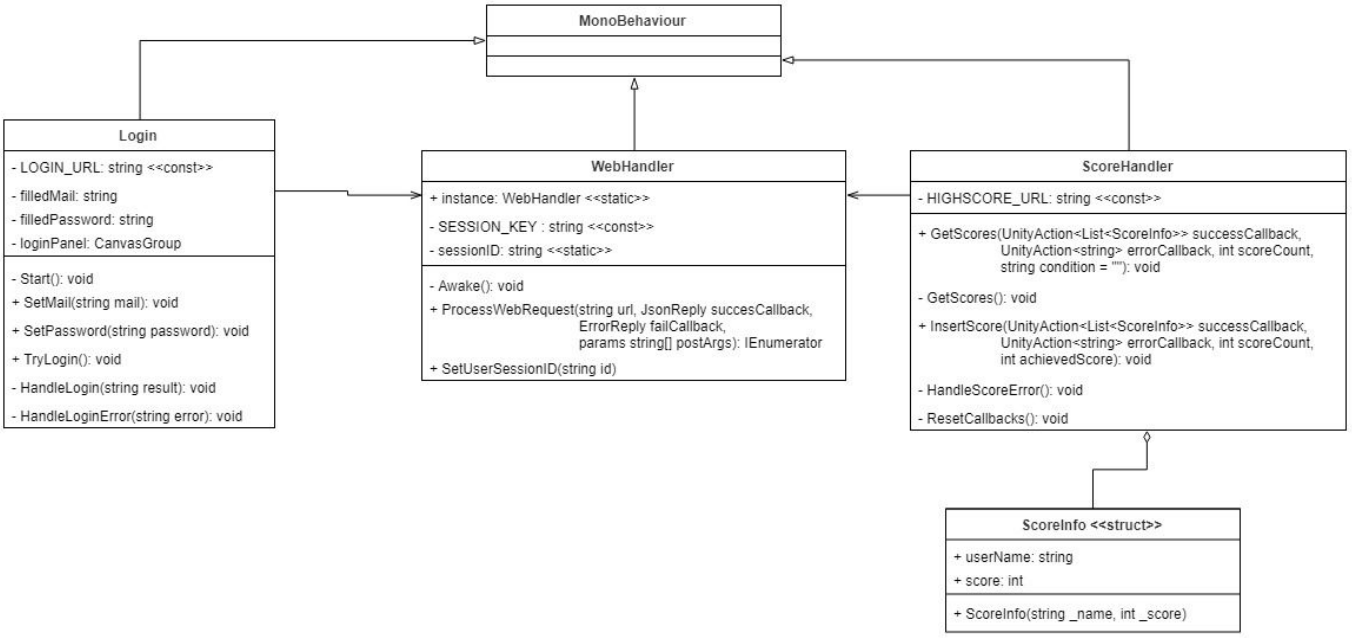
```
//only called on the server
public void InitializeServerPlayer(Player player){
    players.Add (player);

    if (!player.isLocalPlayer)
        TargetInitializeLocalGameInfo (player.connectionToClient, playerCount);
    else
        InitializeGameInfo (playerCount);

    playerCount++;

    if (playerCount == requiredPlayerCount) {
        for (int i = 0; i < players.Count; i ++) {
            players [i].RpcInitializeRaft (i, colors[i]);
            players [i].userInfo.ServerSetUserColor (colors [i]);
        }

        //server always begins (lazy solution):
        activePlayerIndex = players.Count - 1;
        ServerNextTurn ();
    }
}
```

Deze spelers-list wordt gebruikt door de server om aan de spelers te laten weten wie er aan de beurt is. Als een speler een actie gedaan heeft laat hij dit weten aan de server, en dan is de volgende speler van de list aan de beurt. Door middel van een ClientRPC op het juiste speler-object wordt aan de speler laten weten dat hij aan de beurt is.

In een Class Diagram ziet dit systeem er als volgt uit:



Een andere uitdaging was om de scores op een goede manier op te halen uit de databases, en te zorgen dat dit systeem ook in zekere zin modulair was opgebouwd. Ik heb al wel eerder met web requests vanuit Unity gewerkt, maar heb van de lessen wel geleerd hoe ik dit op een nettere manier vorm kan geven. Ook had ik nog nooit eerder met JSON gewerkt, en heb gemerkt dat dit super handig is voor het versturen van data tussen web en Unity. In UML ziet dit systeem er ongeveer als volgt uit:



Conclusie

Vreemd genoeg merkte ik bij veel klasgenoten dat ze op UNET juist behoorlijk vast liepen terwijl ik het zelf wel fijn vond werken. Dit zou ermee te maken kunnen hebben met dat ik het eerder al eens gebruikt had, in ieder geval bevalt het mij wel goed.

De grootste uitdaging van dit project was om de TurnManager op te zetten. Dit komt omdat dit het script is wat de basis vormt van het hele multiplayer systeem. Hiermee worden de spelers in het spel in feite met elkaar verbonden.

Al met al heb ik veel geleerd van dit project: ik heb geleerd hoe ik UNET en Unity werkend kan krijgen, hoe ik een turnmanager kan maken en hoe ik op een nette manier met een database communiceer vanuit Unity.