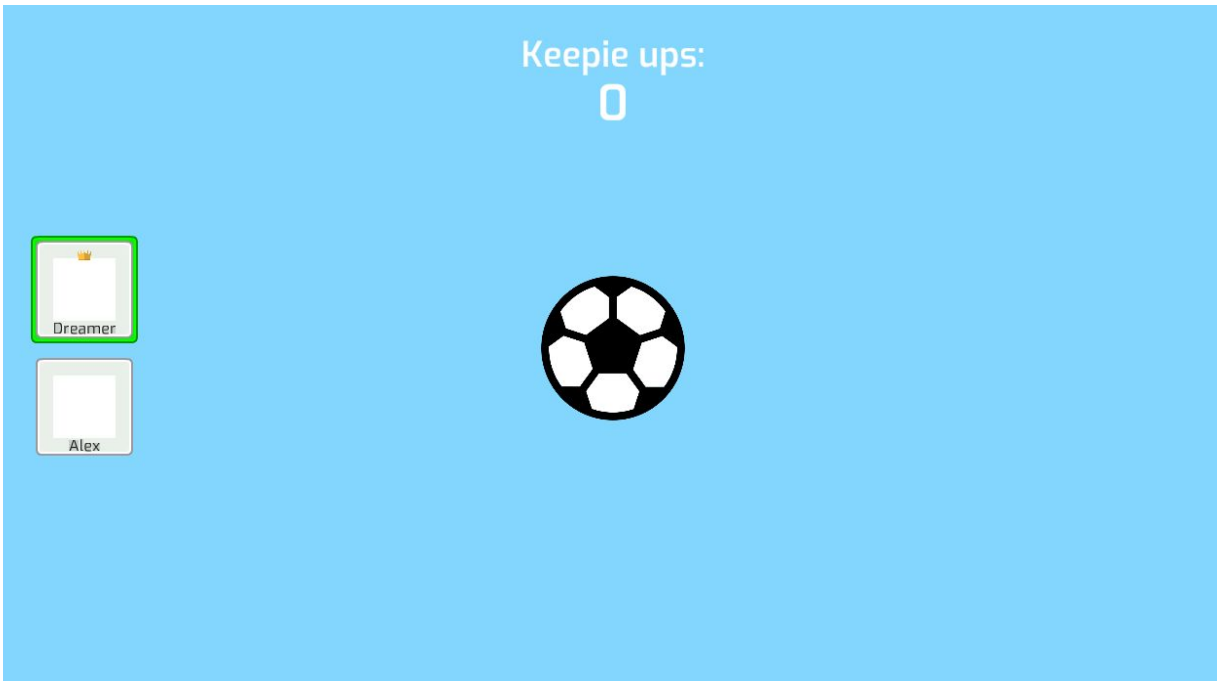
KERNMODULE GAME DEVELOPMENT 4 (MULTIPLAYER)
ALEX VAN DER LINDEN

# KEEPIE UPS
# TURN BASED MULTIPLAYER GAME




## Inleiding

'Keepie Ups' is een *turn based* multiplayer spel waarbij de spelers zo lang mogelijk een bal in de lucht proberen te houden. Door op de bal te klikken wordt er op elke *client* een physics simulatie gedaan die nagenoeg hetzelfde is bij elke gebruiker. Als de bal buiten het scherm komt stopt het spel en kan iedere gebruiker kiezen een score op te slaan.

## Netwerk

Voor het spel heb ik gekozen voor Photon. Photon biedt verschillende opties voor het maken van een multiplayer spel, waaronder PUN (Photon Unity Networking). Hierin is alleen geen mogelijkheid tot P2P (peer to peer) waardoor je per se via de cloud van Photon moet verbinden wat tot kleine extra vertragingen kan leiden.

## Spel

In eerste instantie had ik een variant op Flappy Bird gemaakt, maar door de zwakke basis van mijn geschreven code (en het moeten herkansen van het vak - niet gerelateerd aan dit) wilde ik iets anders maken.

Ik begon met het maken van een schuifpuzzel, waar iedere speler om de beurt een tegel moet verplaatsen. Dit bracht verschillende problemen met zich mee die duidelijk werden tijdens een test. Spelers konden een eigen foto uploaden (via een `byte[]`) om als puzzel te gebruiken, maar als deze groter was dan 500KB werd iedereen uit de wachtruimte gegooid. Tijdens het spelen van het spel moest de indeling van de gehusselde puzzel naar alle clients verzonden worden, wat (afhankelijk van de grootte van de 2D array) voor grote vertragingen zorgde.

Uiteindelijk heb ik gekozen voor het spel 'Keepie Ups'. Dit is een spel wat ik jaren geleden speelde als Flash spel op een website als Spele.nl. Bij dit spel is het de bedoeling dat de spelers om de beurt op de bal klikken om deze zo in de lucht te houden. Als de bal buiten het scherm komt, stopt het spel en kan iedere speler kiezen om een score te uploaden.

## 'Inloggen'

Voordat de speler verbindt met de Photon Cloud Service moet hij of zij 'inloggen' (`NamePopup.cs`). Door een gebruikersnaam en wachtwoord naar een .php script te sturen krijgt de speler vervolgens een *token* die gebruikt kan worden om een score op te sturen. Dit voorkomt dat de speler een score van iemand anders zomaar kan updaten. Het voorkomt niet dat een speler een willekeurige score op kan sturen (d.m.v. onderscheppen van de .php call via bijv. Charles Proxy), omdat de score op de client zelf berekend wordt.

## Verbinden van spelers

Iedere speler kan zelf een *Room* aanmaken (`CreateView.cs`). Hierin kan de speler aangeven met hoeveel spelers er maximaal gespeeld kan worden (in dit geval twee of vier). Deze *Room* wordt dan weergegeven (`BrowserListEntry.cs`) in een lijst (`BrowserView.cs`) die andere spelers kunnen zien. De room kan vervolgens worden binnengegaan via `PhotonNetwork.JoinRoom(string roomName)`.

## Volgorde bepalen

Als het spel gestart wordt gaat iedereen naar een nieuwe scene via `PhotonNetwork.LoadLevel(string sceneName)`. Alleen degene die de room gemaakt heeft (de *Host*: `PhotonNetwork.IsMasterClient`) kan het spel starten. Mocht de host disconnecten dan wordt een willekeurige nieuwe aangewezen door Photon.

Voordat iedereen naar een nieuwe scene gaat wordt de volgorde van beurten bepaald. De volgorde is datgene wat `PhotonNetwork.PlayerList` op dat moment terug geeft. Deze kan je eventueel husselen, maar is niet per se nodig. Deze data is op te slaan in de opties van de room: `PhotonNetwork.CurrentRoom.CustomProperties`.

Eenmaal in de nieuwe scene stuurt iedere client een *Event* (`PhotonNetwork.RaiseEvent`) naar de host. Zo weet de host dat iedereen zeker weten in het spel zit. Vervolgens wordt de bal in de scene gezet d.m.v. `PhotonNetwork.InstantiateSceneObject` (`GameManagement.cs`).

## Beurten

Elke keer dat iemand op de bal klikt gaat de beurt naar de volgende speler in de lijst die is opgeslagen in de room. Naast de lijst wordt er ook een getal (index) bijgehouden van wie er aan de beurt is.

### Stappen

1. Speler klikt bal.
2. Lijst (`Player[])` van volgorde wordt opgevraagd samen met de index.
3. Er wordt 1 bij de index opgeteld en deze gaat weer naar 0 mocht het getal groter zijn dan de lengte van de lijst.
4. `Player[index]` is de volgende speler die op de bal mag klikken (`photonView.TransferOwnership`).

Mocht een speler disconnecten dan wordt deze methode ook toegepast om de lijst te updaten.

## Conclusie

Voor een volgend multiplayer spel wil ik verder kijken dan alleen Photon. Hoewel het makkelijk te gebruiken is, zijn er ook een aantal problemen waarvan ik mij afvraag of andere services dat ook hebben (bijvoorbeeld geen P2P en wachtwoorden op *Rooms*).

## Verbeteringen

- De mogelijkheid om het spel dynamisch in te stellen door bijvoorbeeld opties toe te voegen die de bal een andere grootte geven, de snelheid verandert of de volgorde van de spelers telkens verandert.
- Het opnieuw starten van een spel.
- Een speler in een Room moet de status van zichzelf kunnen veranderen naar ‘ready’ of ‘not ready’.

## Opmerkingen

- Nadat een spel is afgelopen, moet de speler opnieuw ‘inloggen’ en verbinden met de Photon Cloud Services.