**DESIGN DOC I**

**<ARMAMENT>**

****

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Author | Revision Date | Comments |
| 1.0 | Ben Mankin | Feb. 18, 2019 | initiated |
| 1.1 | Andrew Pitt | Feb 21, 2019 | General reqs, logo, updated system overview |
| 1.2 | Alex Cohn | Feb 21, 2019 | Diagrams, descriptions, resource material, other edits |
| 1.3 | Alex Cohn | Feb 22, 2019 | Outline for document, client component description |
| 1.4 | Keith Bosworth | Feb 22, 2019 | Add Use Case Diagrams |
| 1.5 | Jeremy Scott | Feb 22, 2019 | Added to Client Component Classes, helped create diagrams |
| 1.6 | Keith Bosworth | Feb 22, 2019 | Added ClientComponent, PhotonCloudComponent Diagrams |
| 1.7 | Jeremy Scott | Feb 22, 2019 | Minor edits/formatting |
| 1.8 | Alex, Andrew, Keith | Feb 22, 2019 | Edits and formatting |
| 2.0 | Andrew Pitt | April 22, 2019 | Edits based on latest Δs |
| 2.1 | Keith Bosworth | April 28, 2019 | Update System Overview and use case user flow diagrams |
| 2.2 | Jeremy Scott | April 28, 2019 | New Sequence Diagrams |

**Table of Contents**

[**System Overview**](#_f52r8zkyds56) **4**

[**References**](#_7wrjhe45tcti) **5**

[**Document Overview**](#_pjxlinpbr1dg) **5**

**Glossary**

[**Software Block Diagram**](#_myn0opv2rk4m) **6**

[**Software Block Diagram Description**](#_yogzx6h2nwic) **6**

[**Class Diagrams**](#_wt90og3eghtw) **7**

[Client Component Class Diagram](#_hf6mxyrwq77k) 7

[Client Component Classes Description](#_haqf1y9svu96) 8

[**Use Case Sequence Diagrams**](#_ezf7dqwptefn) **10**

[UC-1: Random Matchmaking](#_4taz0bslqssm) 10

[UC-2: Hosted Match](#_uz42cq8u7uqy) 11

[UC-3: Settings Management](#_tzmzmneka43k) 12

[UC-4: Tutorial](#_1sa3y1wwuoum) 13

[UC-5: Leaving Game](#_5guwspizxek) 14

## System Overview

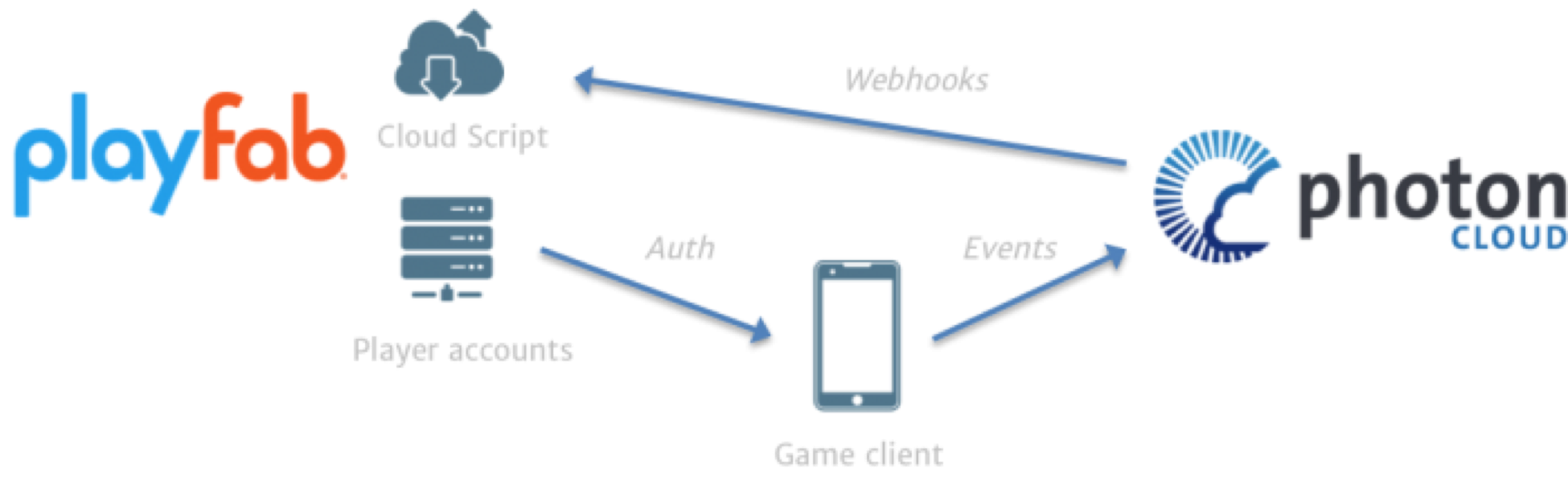
This section provides an overview of the Armament project: an original, networked, multiplayer, team-based video game built for PC, Mac, iOS, and Android platforms. Armament is built with the Unity game engine and API, Photon Networking servers and API, and PlayFab backend database and API.

Gameplay will consist of two stages played in succession: first, an *Armament* stage where players gather weapons and resources, and subsequently, a *Battle* stage where combatants fight for control of the arena using the resources they’ve acquired. Original sounds and art will be created for the project in addition to existing assets.

Armament will be designed and driven with the Unity engine and API, which contains numerous scripts and libraries that provide abstraction for the low-level details of physics rendering, graphics processing, animation, A.I, platform-specific builds, and system analytics.,

Player information is stored in a database provided by PlayFab. Users can register accounts and authenticate from both PC and mobile devices. Once logged in, players will see the Launcher where they can choose to play a game, or they can check the statistics stored in the database, which may be accessed through the leaderboard. If they choose to play, their statistics during that game will be updated to the leaderboard. Players can also add friends through the Launcher, which will allow them to invite those friends to private games. This feature gives players the ability to stay in touch with players they enjoyed playing with.

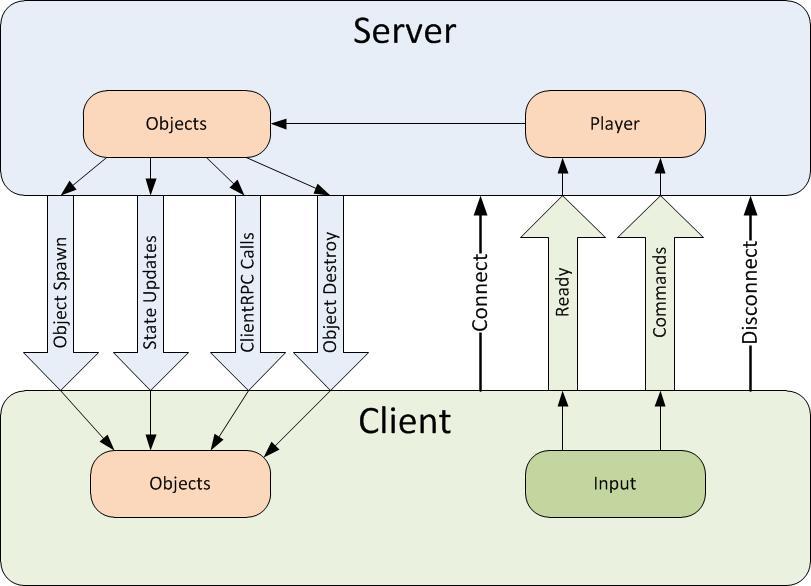
The interaction between Unity clients, the Photon cloud, and PlayFab can be seen below in the abstraction provided by **Figure 1.**

****

**Figure 1.** General overview of Unity client integration with Photon and PlayFab from a mobile perspective (similar for PC).

To play the role of the serving or host device, one client will assume the role of master client, which will synchronize remote clients by broadcasting RPC through Photon servers. A master client is a specialized type of local client that synchronizes remote clients using Photon servers as a means of communication, which sends and receives changes in specific game state based on client input. This implementation is headless in that it only passes RPC calls and choice data through Photon servers, as opposed to processing graphics and physics data within the Photon server space.

In order for Armament to synchronize across every connected device, copies of each object created in the game will exist in memory on both the master and remote clients. The master client device assumes the role of the authority to keep track of changes made to various GameObjects, as well as communicating those changes to the clients. The client/server relationship, RPC calls, data flow, GameObject storage, and state changes can be seen in **figure 2** below:



**Figure 2.** Overall data flow, object storage, and state changes from server (master client) to remote and local clients.

Data will be broadcasted from client to client using RCP and other modes of communication and data synchronization via the Photon server, and through the internet via UDP and TCP. The Photon networking framework provides a robust networking API built specifically for Unity projects in order to meet networking requirements. Clients send input, which is received by listening to various RPC events (for example OnMouseClick() called from within an RPC wrapper to listen to mouse clicks supplied from user input). The Photon Networking API provides abstraction of low-level socket code. **Figure 3** describes the user flow to begin playing online with other players.



**Figure 3.** User flow to begin playing Armament

When joining a game, players must use the in-game menu to connect to a Photon *name server*, which gives them access to a *master server*. Master servers are geographically located around the globe to provide low ping times to all clients, regardless of their location. Master clients will then place the clients in a *master server for matchmaking purposes.* . When a player finds a match they will communicate with other clients via a Game Server that is responsible for hosting the game room they are in.

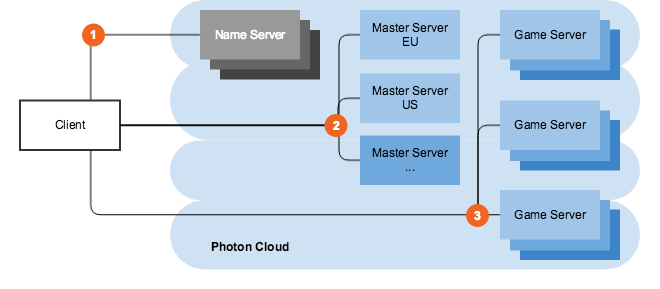
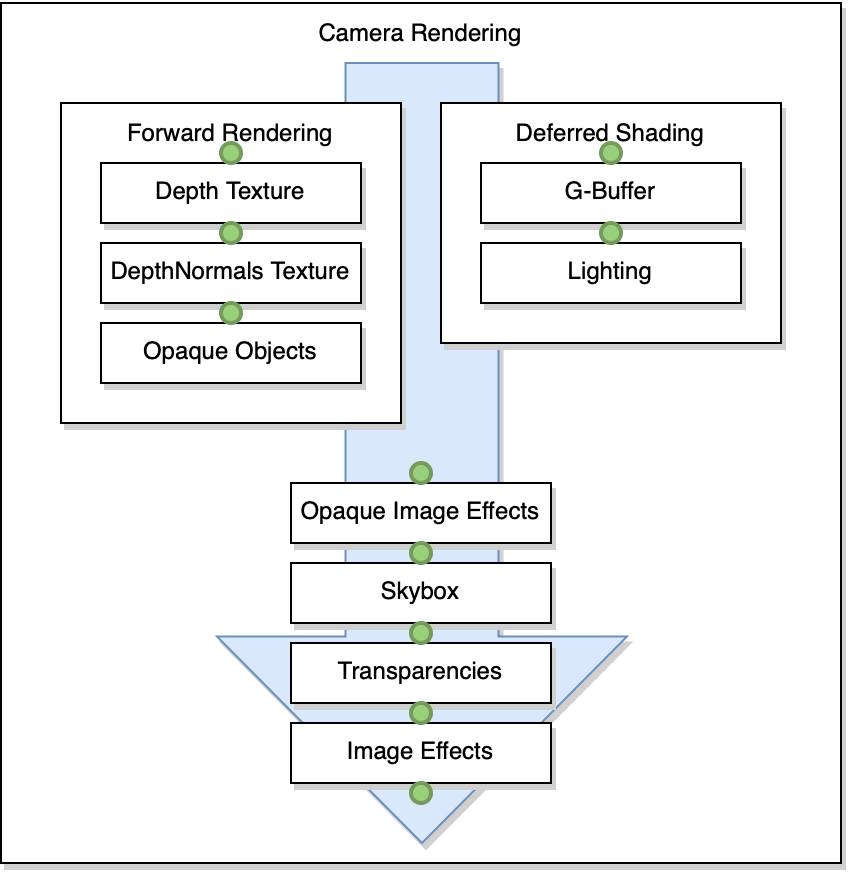


Figure 4.Master server and game server layout..

In the case that the host connection quality terminates or becomes suboptimal, the Photon API offers a host migration service which is called in order to move server identity to the next available device. Sending RPC through the Photon server adds one layer of security for clients in that the IPs of remote clients are managed by the Photon server instead of seen directly by the master.

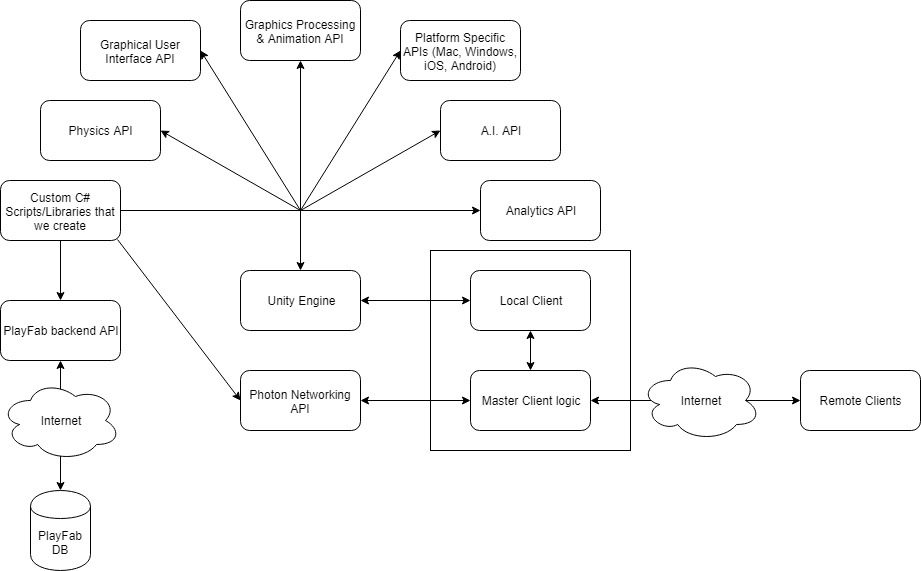
## The onus of graphics processing is placed on the client devices, which may flex the master client’s processing capabilities somewhat, but overall is not an impediment considering how picky our master client is in terms of choosing what to broadcast to clients through Photon servers. On the client side, the Unity engine will render a camera placed in the virtual environment, resulting in the data flow represented in figure 5 below. After graphics processing, certain state changes to the GUI will be communicated as attributes via various synchronization mechanisms provided by photon to all clients over the Internet.



**Figure 5**. Camera rendering data flow

## During gameplay, players have the option to toggle an A.I. controller. When this controller activated the game automatically takes control over the player avatar’s movement and actions. If the A.I. controller is activated during the *Armament* stage, the A.I. player will target (i.e., go to) known gun spawn points in order to pick up guns. Along the way, the A.I. may recognize that a gun, which it is not currently targeting for pickup and not yet picked up by another player, has come into view. . When it sees a valid gun target, it will run towards it and attempt to pick it up. During the Battle stage, the A.I. will wait for opponents (i.e., players on the other team) to come into view or shoot at it. Either event will trigger the A.I. player to target the opponent. The A.I. player will pursue its targeted opponent even if the player tries to run away. If the A.I. player gets the target in its crosshairs, it will immediately shoot..The A.I. player always calculates shortest path to its destination whether or not its target’s position changes. A high-level representation of the interlocking systems is described below in figure 4.

## System Block Diagram



**Figure 4.** A high-level view of the components in Armament.

## 

## Glossary

* **Master client**: the client that is designated to act as a pseudo server for all other clients. The master client becomes responsible for making decisions and coordinating actions that would typically be the responsibility of a server in a server-client model. Any client that joins a game room can potentially become the master client at some point. By default, the master client is chosen in the order of who entered the game room first.
* **Remote client**: all clients that are not currently the master client.
* Name server: the first server that every client contacts, which provides the list of available regions.
* **Master server**: every region has a completely separate master server for matchmaking.Game server: hosts game rooms
* **Launcher**: the first scene presented to the user upon starting the game. In this scene, a user has the ability to log in to their account, choose their gameplay options, and enter a game room to begin playing the game.
* **GameObject**:
* **Prefab**: a “prefabricated” set of GameObjects that are linked in a parent-child relationship. Prefabs have components that affect the way they behave (as it appears to the user) after graphics processing.
* **Components**: scripts which can be added to GameObjects in order to change the GameObject behavior, appearance, and properties. The Unity API contains certain fundamental components that can be added to a project. Developers also have the capability to write original scripts as components.

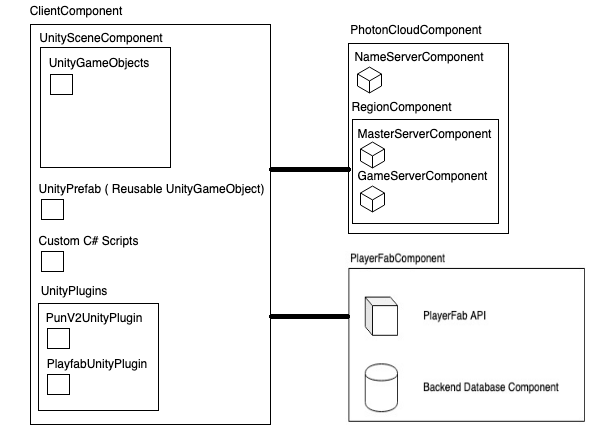
## Document Overview

The software design document describes the software architecture and how the requirements are mapped into the design. This document will be a combination of diagrams and text that describes what the diagrams are showing.

It includes the following sections:

* Client Component Classes
* Class Diagrams
* Sequence Diagrams

## Software Block Diagram

****

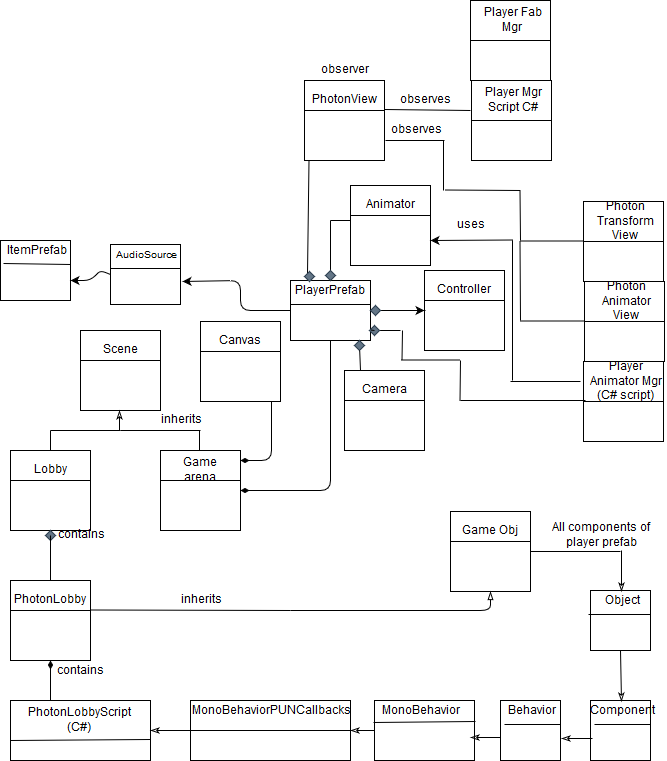
**Figure 4.** System block diagram.

## Software Block Diagram Description

**Figure 4** shows the System Block diagram for Armament. The Photon Cloud and Playfab components will interface over the internet with the client component via Unity plugins. These components will provide the client with user authentication, game synchronization, and host-migration services.

## Class Diagrams

### Client Component Class Diagram

**Figure 5.** Client Component Class Diagram.

### Client Component Classes Description

This section describes the diagram in **Figure 5** in more detail.

* **Unity Scenes**
  + **Lobby Scene** (Lobby.unity)
    - First unity scene a user sees
    - This is the scene where the user can choose to join a random existing room or create a new room if one is not available
    - This scene will contain a GameObject (invisible to the user) called Photon Lobby with the Photon Lobby Script attached to it
  + **Game Arena Scene** (GameArena.unity)
    - Initially, we will only provide one Game Arena unity scene. In the future there may be more.
    - This is where the user will play the game
    - Contained GameObjects:
      * **Scene Props GameObject**
        + a collection of game objects that represent the visual objects in the arena (e.g., the ground, boxes, ramps, etc…)
      * **Canvas GameObject**
        + the parent game object of all 2D UI components (e.g., information displays and on-screen controls)
      * **Game Manager Prefab**
        + described in Unity Prefabs section below
* **Unity Prefabs (reusable Unity GameObjects)**
  + **Game Manager Prefab**
    - One per Game Arena scene
    - It is a prefab in case our game has multiple Game Arena scenes in the future, in which case it can be reused in those scenes
    - Attached Components:
      * Game Manager (Script)
        + described below
  + **Player Prefab**
    - Represents a player character
    - Used by PUN to instantiate player character on all clients via call to PhotonNetwork.Instantiate()
    - Attached Components:
      * **Animator**
        + binds Unity's animation system to a GameObject
      * **Character Controller**
        + used for third-person or first-person player control that does not make use of Rigidbody physics
      * **Audio Source**
        + plays back an audio clip in the scene
      * **Photon View (Script)**
        + PUN's NetworkView replacement class for networking
      * **Photon Transform View (Script)**
        + synchronizes position, rotation and scale of a GameObject
      * **Photon Animator View (Script)**
        + synchronizes Mecanim animations
      * **Player Animator Manager (Script)**
        + described below
      * **First Person Controller (Script)**
        + described below
      * **Player Manager (Script)**
        + described below
* **C# Scripts**
  + **Photon Lobby Script** (PhotonLobby.cs)
    - Purpose: setting up connections between the player and the photon servers and either allow the players to join an existing room or create a new room if one is not available
    - Extends: MonoBehaviourPunCallbacks
    - Uses: Photon.Pun, Photon.Realtime
  + **Game Manager Script** (GameManager.cs)
    - Purpose: responsible for watching for player connections and loading objects into game arena
    - Extends: MonoBehaviourPunCallbacks
    - Uses: Photon.Pun, Photon.Realtime
  + **Player Manager Script** (PlayerManager.cs)
    - Purpose: keeps track of player health, resources, and communicating player actions via photon network
    - Extends: MonoBehaviourPunCallbacks
    - Implements: IPunObservable
    - Uses: Photon.Pun
  + **Player Animator Manager Script** (PlayerAnimatorManager.cs)
    - Purpose: responsible for interacting with the animator component on the Player Prefab
    - Extends: MonoBehaviourPun
    - Uses: Photon.Pun
* **Unity Plugins**
  + **PUN v2 Unity Plugin (Server Interface)**
    - Photon Unity Networking (PUN) is a Unity package for multiplayer games. Flexible matchmaking gets your players into rooms where objects can be synced over the network. RPCs, Custom Properties or "low level" Photon events are just some of the features. The fast and (optionally) reliable communication is done through dedicated Photon server(s), so clients don't need to connect one to one.
  + **PlayFab Unity Plugin (Authentication/Database)**
    - **Authentication**
      * PlayFab is integrated with Photon in order to authenticate users
      * Once a user is authenticated via the PlayFab API, a Photon authentication token is requested
      * User is authenticated in Photon using Photon authentication token and PlayFab values
    - **Database**
      * Users in PlayFab are associated with sets of key-value pairs (presented in JSON format) defined by PlayFab. Setting and getting these values is done through the PlayFab API

## Use Case Sequence Diagrams

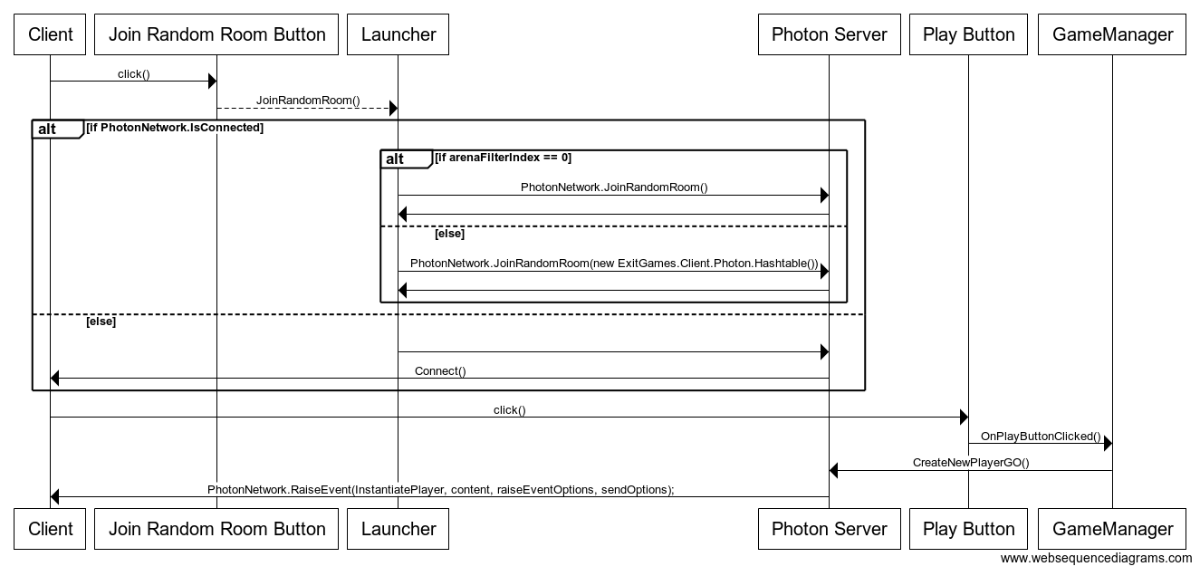
**UC-1: Random Matchmaking**

**Summary:** A user wants to find a live match along with other users who are also looking for a match

**Preconditions:** User is logged in and has navigated to “Matchmaking” in Launcher

**User Actions:**

1. User selects “Join Random” option
2. User waits in scene view until they are ready to play
3. User selects “Play Game” button

**Diagram: **

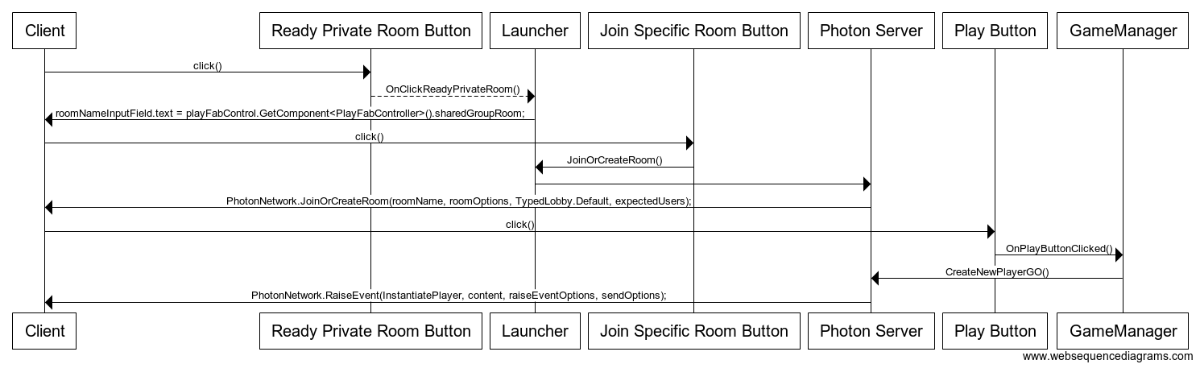
**UC-2: Private Match**

**Summary:** A user wants to be placed into a live match with specific users.

**Preconditions:** User is logged in and has navigated to “Matchmaking” in Launcher. A private room has been created, and shared with the user.

**User Actions:**

1. User selects “Ready Private Room” option
2. User selects “Join Specific Room” option after room name populates
3. User waits in scene view until they are ready to play
4. User selects “Play Game” button

**Diagram: **

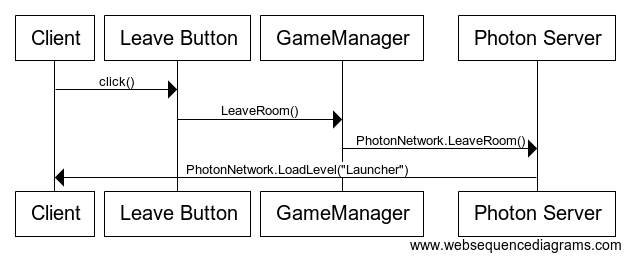
**UC-3: Leaving Game**

**Summary:** A user wants to leave a match they are currently playing.

**Preconditions:** User is logged in and is currently in a match.

**User Actions:**

1. User selects “Leave Game” option

**Diagram: **

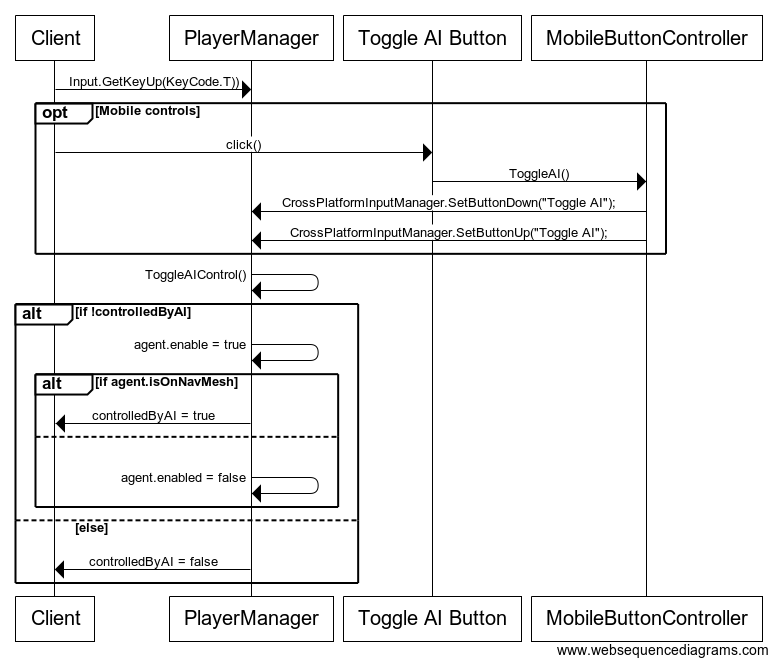
**UC-4: AI Toggle**

**Summary:** A user enables the game to control their avatar.

**Preconditions:** User is logged in and is currently in a match.

**User Actions:**

1. User presses “T” to toggle AI on
   1. In mobile the player presses “AI” button

**Diagram: **

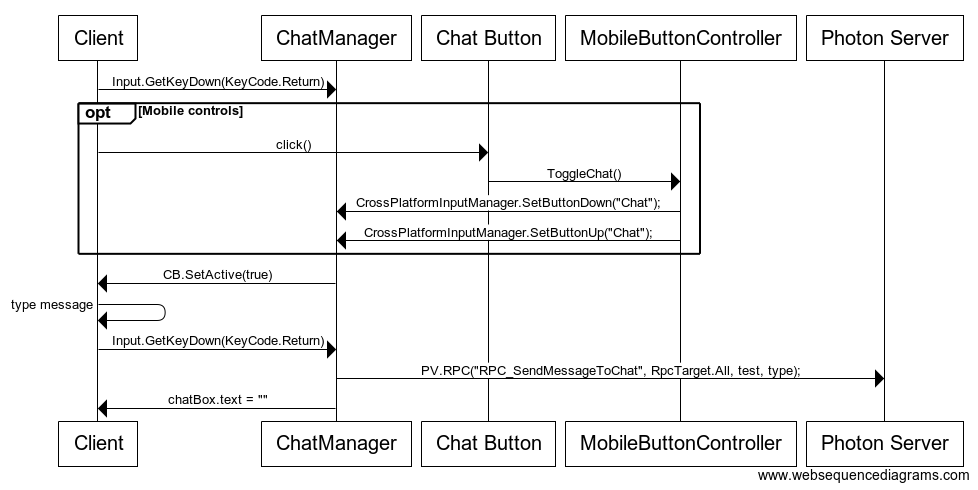
**UC-5: In-Game Chat**

**Summary:** A user sends a message to players in the current match.

**Preconditions:** User is logged in and is currently in a match.

**User Actions:**

1. User presses “Enter” key to initiate chat box view
   1. In mobile the player presses “Chat” button
2. User types a message into chat box
3. User again presses “Enter” key to send message

**Diagram**

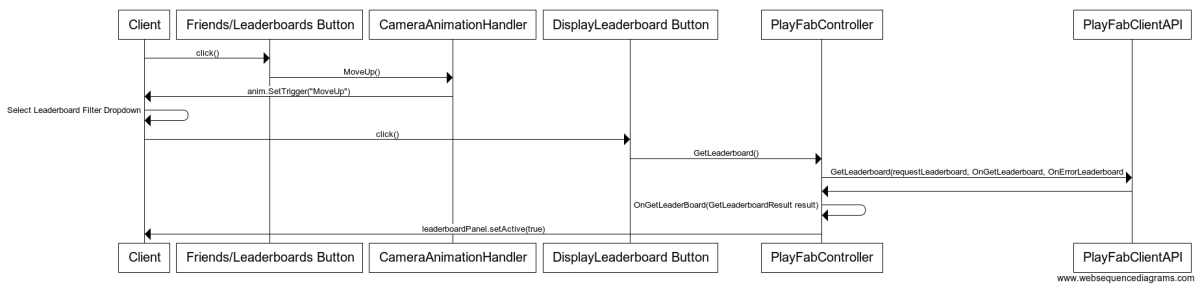
**UC-6: View Leaderboards**

**Summary:** A user views leaderboards and compares global stats of players

**Preconditions:** User is logged in and in Launcher

**User Actions:**

1. User selects “Friends/Leaderboards” button
2. User selects which leaderboard they want to view from dropdown
3. User selects “Display Leaderboard” button

**Diagram:**  

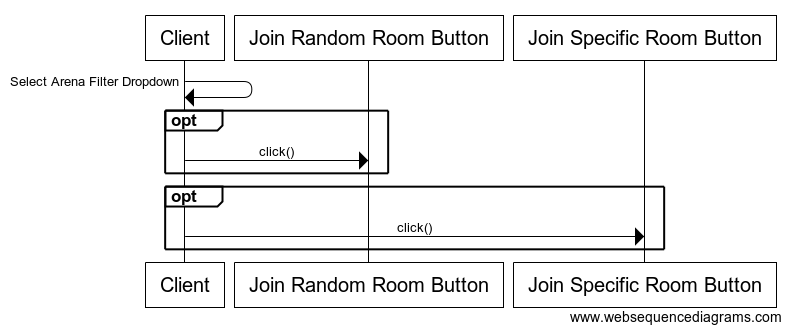
**UC-7: Select Arena**

**Summary:** A user selects an arena to play a match in from list of available arenas

**Preconditions:** User is logged in and in Matchmaking screen

**User Actions:**

1. User clicks “Arena Filter” dropdown
2. User selects an arena from the list
3. User selects either “Join Random Room” or “Join Specific Room”

**Diagram:**  

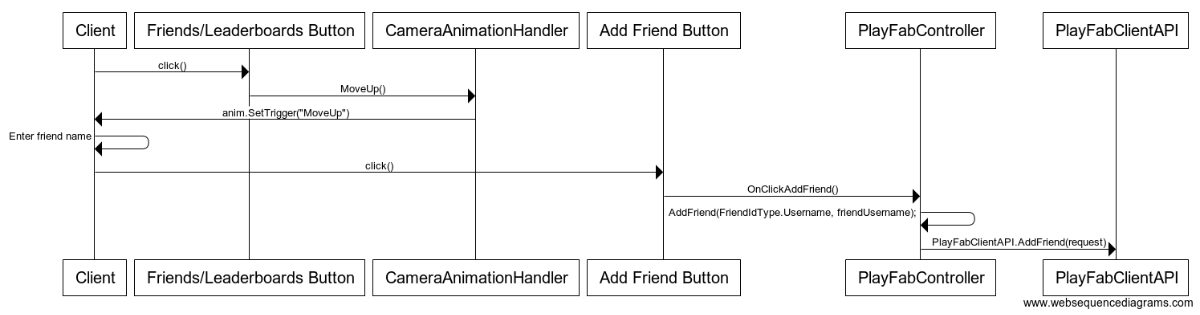
**UC-8: Add Friend**

**Summary:** A user adds a friend to their friend group

**Preconditions:** User is logged in and in Launcher

**User Actions:**

1. User selects “Friends/Leaderboards” button
2. User enters a friends username in “Enter a friend’s username” field
3. User clicks “Add Friend” button

**Diagram:**  

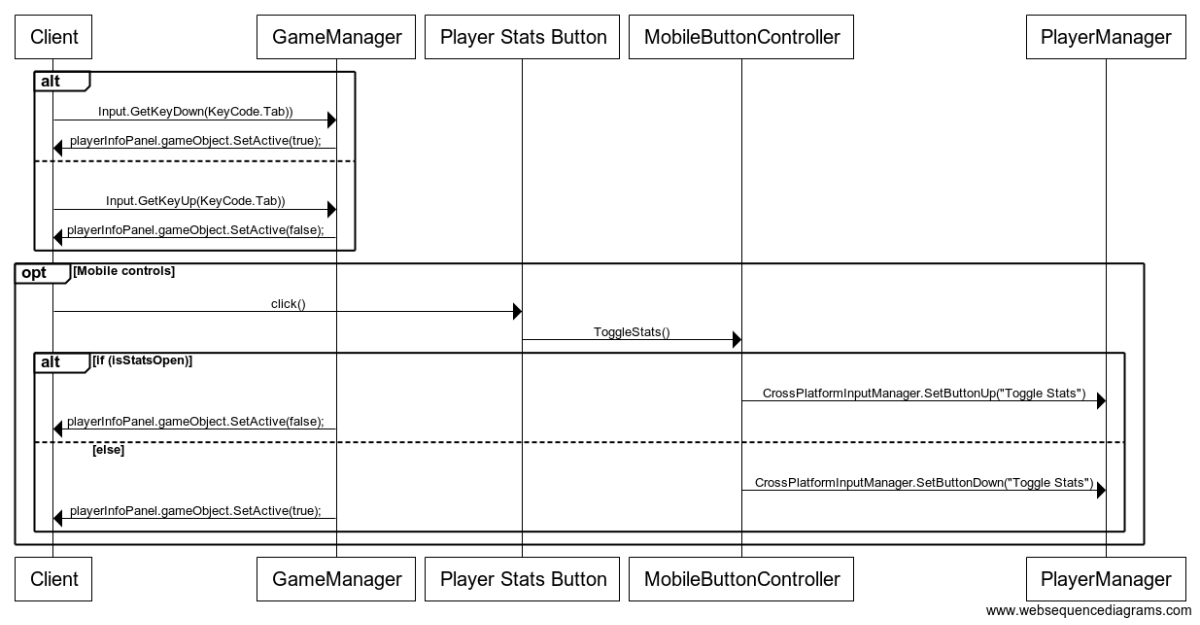
**UC-9: View Player Info in Match**

**Summary:** User is able to view info of other players during a match

**Preconditions:** User is logged in and in a match

**User Actions:**

1. User presses the “Tab” key to bring up player info overlay
   1. On Mobile, the user presses the “Player Stats” button to toggle overlay

**Diagram:**  

**UC-10: Team Preference**

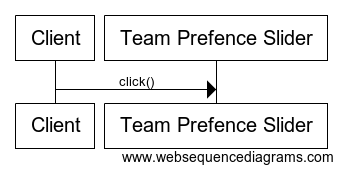
**Summary:** User selects a team preference for when they join a match

**Preconditions:** User is logged in and on Matchmaking screen

**User Actions:**

1. User uses team preference slider to indicate preference.

**Diagram:**



## References

* Unity user Manual 2018.3 <https://docs.unity3d.com/Manual/index.html>
* Photon documentation <https://doc.photonengine.com/en-us/pun/v2/getting-started/pun-intro>
* Playfab documentation <https://api.playfab.com/docs/general-getting-started>
* Git branching model https://nvie.com/posts/a-successful-git-branching-model/

### 

### 