Overview

The system has seven components. Four of the seven components are in client side as an unity application including two UI components for representing gameplay and other information that can be accessed in menu and two components for handling the input from GUIs for game and other places like menu and sending feedback to view(GUI) to update it. The other three components are implemented on server. Specifically, both Server Game Component and Server Other Information Component are implemented on photon which is a unity network. The database component works with the server components, makes the queries and accesses to the database.

# Definitions

* Client: The software running on mobile devices.
* Client Game Component: The software component implemented on the client that receive data from server game component to update game objects and reflect them to UI Game Component
* Client UI Game Component: The software component implemented on the client that handles user action in game and update through implementation of a GUI and update game objects using the data received by client game component
* Client Menu function Component: The software component implemented on client that gets data like user information, highest score, friend’s information and etc from the server and show them on GUI component
* Client Menu GUI component: The software component implemented on the client that handles user request in GUI and display information received from client menu function component
* Server:The software running on internet-facing machines hosted on pun(photon unity network)
* Server Game Information Component: The component of server that stores the data about the game objects, update them upon user’s request from client game component and send back to client game component
* Server Other Information Component: The component of server that handles data other than the game, for example, user’s detail, user’s friend and lobby data. It authenticates user’s identity and gets user data from database and sends back to client.
* Database: The database component work within the server. It connects to the mysql database. The database implementation application have database access objects making queries according to requirements. And the create user and message objects are created by the database access objects and passed to the server.

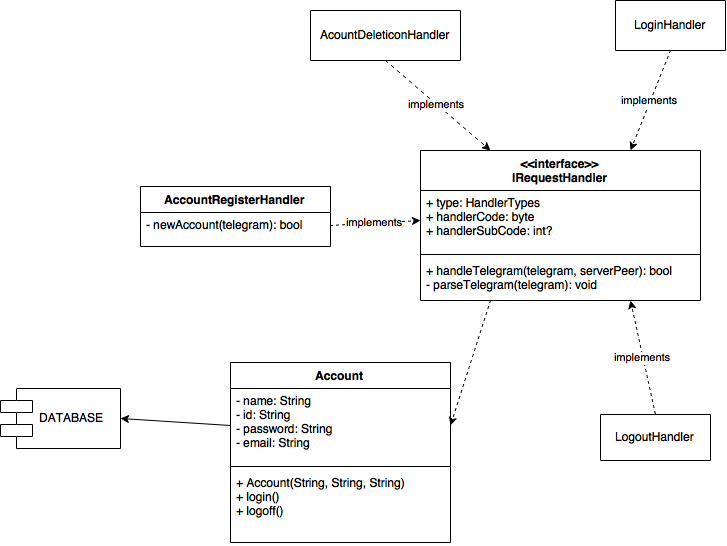
# Architectural Structure

Structure.png

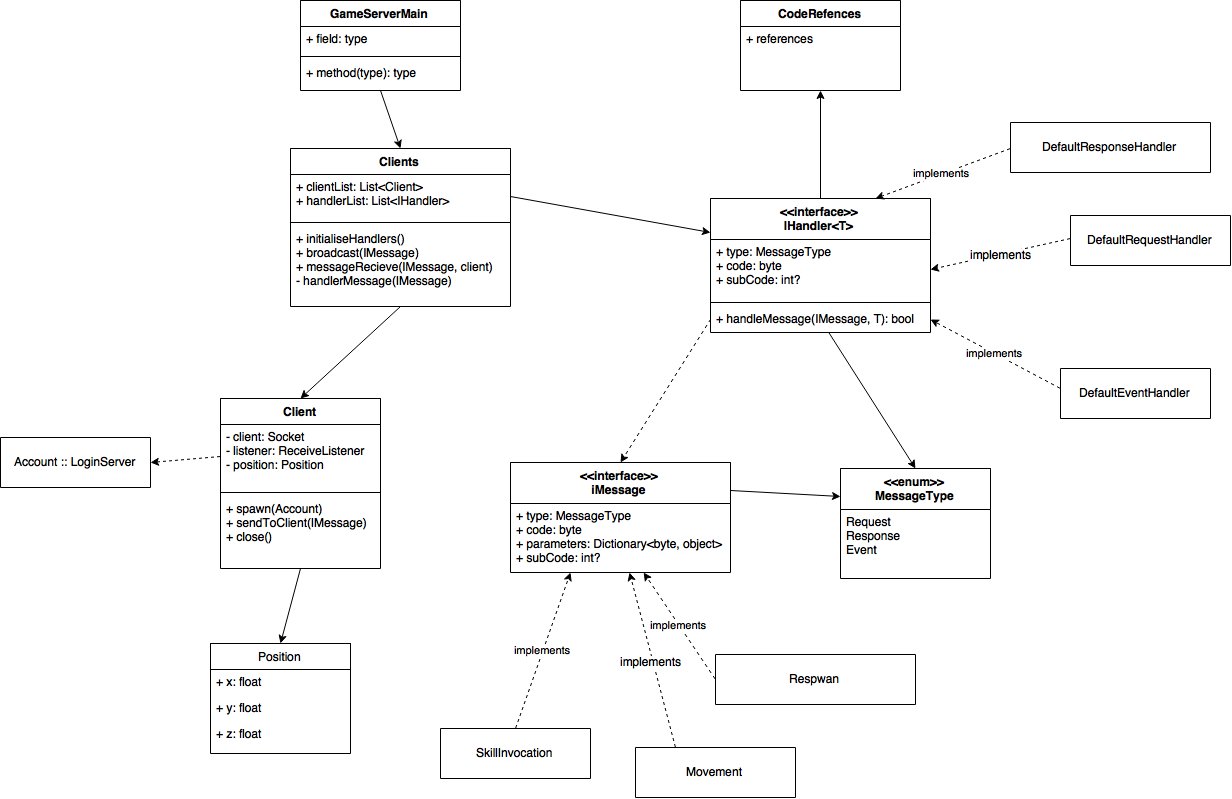
# Server Design

The application server is implemented with C# on Windows server with assist of the Photon Socket Server SDK. Server side consist of two components: login server and game server. Brief design diagrams as follows:

Login Server:



Game Server:



# Client Design

User class: user class is used as a controller in MVC design pattern. It has functions that are called when corresponding buttons in UI are pressed. It can get player’s position and status from the game server and display on the view. The functions in user class include moving, action, logging in, sending chat message in game, updating from server and etc. Java API is used for sending user’s interaction to game server.

Ball class: ball class is the controller of the game object ball. It has update function that update its position according to the data on game server.

Referee class: a referee class that has the ball object and record score for both side. It has functions to determine if one of the team scored and send it to game server.

The other objects are static and implemented in Unity. (The playground may have rotating function)

\*all the data sent to game server is via JAVA API with two arguments which are the message and the user ID.

# Database Component Design

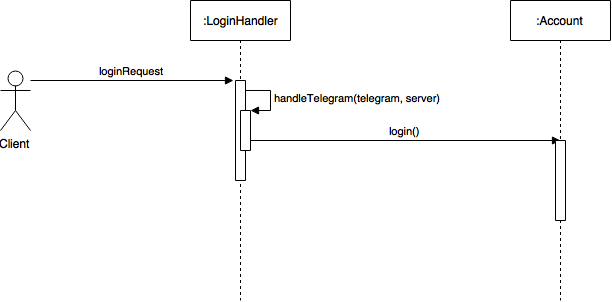
The database component uses mysql on Windows server and is implemented by C# on the server. The database application creates the game objects pass to the server. databaseDesign.png

# Sample Interactions

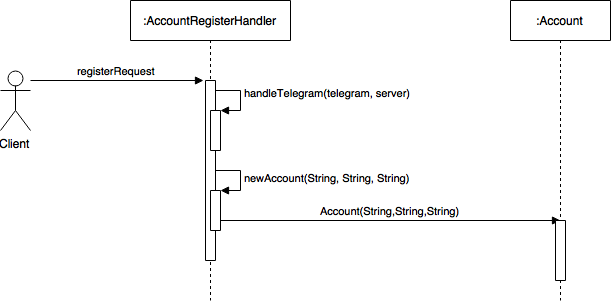
1. Action of a player in game

action of player.png

1. Login

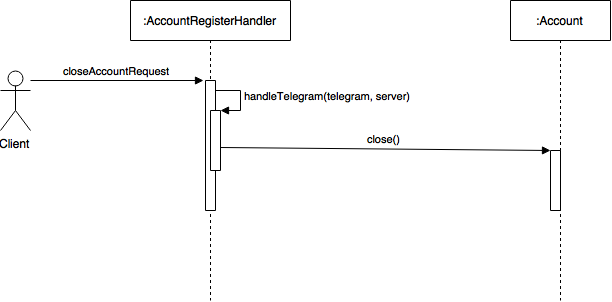


1. Account Creation

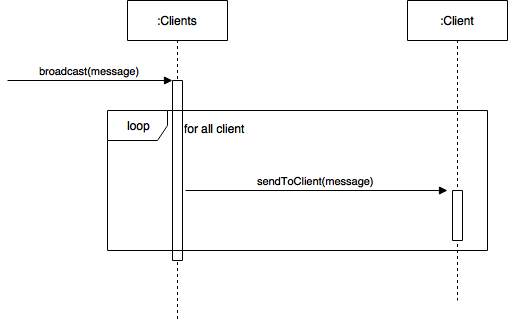


Note: Class Account will interact with database to accomplish relevant task, same applies with all requests about Account.

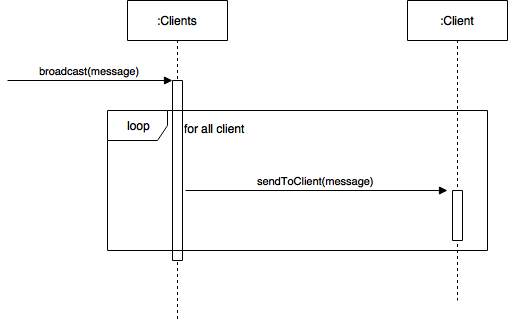
1. Account Deletion



1. Broadcast to clients



1. Handling client message



# Network Interface

* **Server Messaging Component**

The network interface between server and client is via TCP networking protocol and HTTP request.

Sample Message Packet

[]:list // contains a list of objects name

{}: dictionary // contains a list of objects with its effects, such like animations.

Message :{

timestamp: int;

from\_user: int;

to\_user: int;

message:[char]; // message that clients send to server, which may redirect to other clients.

pos\_x:float;

post\_y:float;

action:[char];

is\_charging:int // sign of using signature skills

}

Similarly, a POST method is used via HTTP request. Therefore a sample POST request would be:

Message :{

timestamp: 0044;

from\_user: 004;

to\_user: 006;

message:”How u doing today?”;

pos\_x:37.111;

post\_y:15.215;

action:”up”;

is\_charging:1 // sign of using signature skills

}

As the server received the message from any clients, it will issue a signal to indicate client that message has been received. One typical feedback would be:

signal:[char];

signal = “Message Accepted”;

* **Client Messaging Component**

The message sent from client is via JAVA API from user’s mobile side.

/\*\*

\* this function takes two arguments and encode the message and sends it to the server

\* @message, the message that is going to send to server

\* @userID, identifies which user is sending this message

\*/

Public static void sendMessage(String message, int userID);

/\*\*

\* this function takes two arguments and decode the message and receives message from

\* server

\* @message, the message that is going to send to server

\* @userID, identifies which user is sending this message

\*/

Public static void receiveMessage(String message, int userID);

/\*\*

\* this function takes two arguments and sends actions that users used to perform their action

\* @action, the actions that users want to perform at the moment of hitting any button

\* @userID, identifies which user is sending this message

\*/

Public static void sendAction(Action action, int userID);