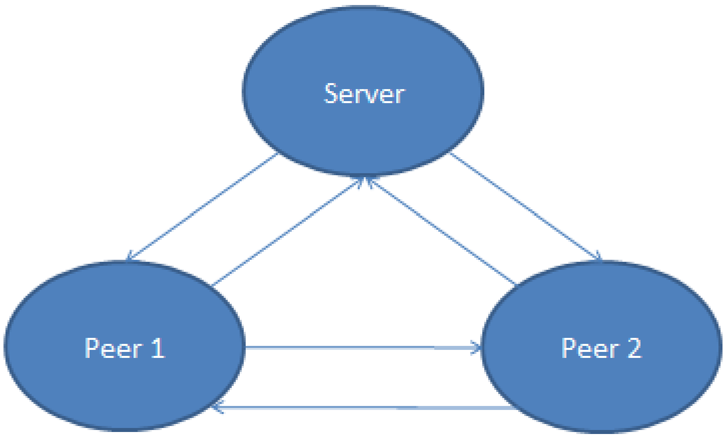
Goal: Our game is written in purpose to understand and improve our skills in P2P network, HTML5 SVG animations, CSS3 flexbox and use third-party libraries as JQ UI, Peerjs, JQ.

Description: football game designed so to provide possibility for 2 people play football with simplified rules. These rules define every team consists one goalkeeper at a time only no other kickers on the field in the game to be able to control one keeper by one player. Player commands one's goalkeep only from left/right side depending on game connection sequence. Every goalkeeper returns the ball moving by it-self. The motion is constant. Acceleration is absent. Motion direction changes every 'hit' field border or a keeper.

Web game exterior architecture

In our web application we used P2P server and library from peerjs.com. Free P2P service at peerjs.com provide access to message, video and data exchange.Peer server is a connections broker, PeerJS connects to a PeerServer.No peer-to-peer data goes through the server. It doesn't provide secure connection.



Data transfer types:

Exchanging data types	Data's type
Player position	Boolean
Ball position	Num
Goal count	Num
Chat messages	String
Restart	Boolean/string
Undefined data	any

Football game function/action list

Function()	Action
	Connection Service Block
startNewGame	get peer ID, wait and connect guest peer;
connectToGame	invoke dialog window and connect to host peer;
connectToPeer	get peer ID, connect to host peer;
dataListener	listens to incoming data and redirecting to proper functions;
checkReadyStatus	get peer ID, wait for incoming peer connection;
startGameAfterCheck	handles dialog windows, invoke start game function;
restartRequest	Send request to start game again to second player

Chart Block

sendMessage	Listens to outcoming messages
showInChatScreen	Renders all messages
autoScroll	Message scroll ability

Game Run Block

startGame	sets and invokes physics methods in proper intervals;
ballOutOrGoal	checks ball and goal location intersection and calculate new ball;

Game Run Block

moveBallData	read and render received ball coordinates;
moveBall	calculate and render ball coordinates;
collapsBallKeeper	checks ball and keeper coordinates intersection;
moveKeeper	reads and render goalkeeper coordinates;
ballOutOrGoal	checks ball and goal location intersection and calculate new ball position or goal count;

Use Cases

It is use cases analysis of football game web app designed to provide fun and sport activity by interacting users (hereof players) with each other through internet. Because of it this game as all others applications has its use cases. Use cases represent players and p2p server as actors. Any player has ability to click ‘Start new game’, ‘connect to game’, ‘close connection’ buttons. Also in case of game run user may manipulate one’s goalkeeper, and at any time may send and receive messages in the chat. Detailed use case chart represented on the next page below. The detailing of use case is superficial, detailing style is ‘Fowler style’ to show who is actor, what is main function, how interaction passes.

Error Handling

Our web app like many others may could work unforeseeable)) I am joking, but seriously in JavaScript when an error occurs, script will normally stop and generate an error message. So we also tried to use "try-catch" method to handle possible errors in our program and show them for user in more friendly view.

For example, when you start to type message before creation a connection in our code it generates an error because connection is not yet assigned for a "conn" variable and we can`t send any message. In this case JS throw an exception like:

"TypeError: conn is undefined". So we applyed this method in our function

```
function sendMessage(message){
  try {
    conn.send({type:'chat', msg:message});
    showInChatScreen(false, message);
  }catch(err) {
    showInChatScreen('error', 'You need to establish connection first!');
  }
}
```

Found Bugs

Solved: When connection is established, for start game users must send a confirm request "ReadyToGame" to each other. In this moment one or other request sometimes could not be delivered (just lost during sending - depends on stability of internet connection) .

Solved: Each user is deal with two checkStatus parameters. The first for himself and the second for the connected user. When user click confirmReadyStatus (to set own parameter into ready Status) it sends a request to another user with ask to confirm that he received it. After that User2 sends back answerRequest that he received checkStatus from User1. Only after that User1 can set his own statusCheck on true. In the same way it works on the other side for User2. He clicks -> Sends Request -> gets Answer -> Sets his statusCheck on true.

In this case in modal window user always can see his actual ReadyStatus (the same status can see User on the other side of connection). So in case if statusCheck request was lost, user can simply repeat sending by clicking one more time.

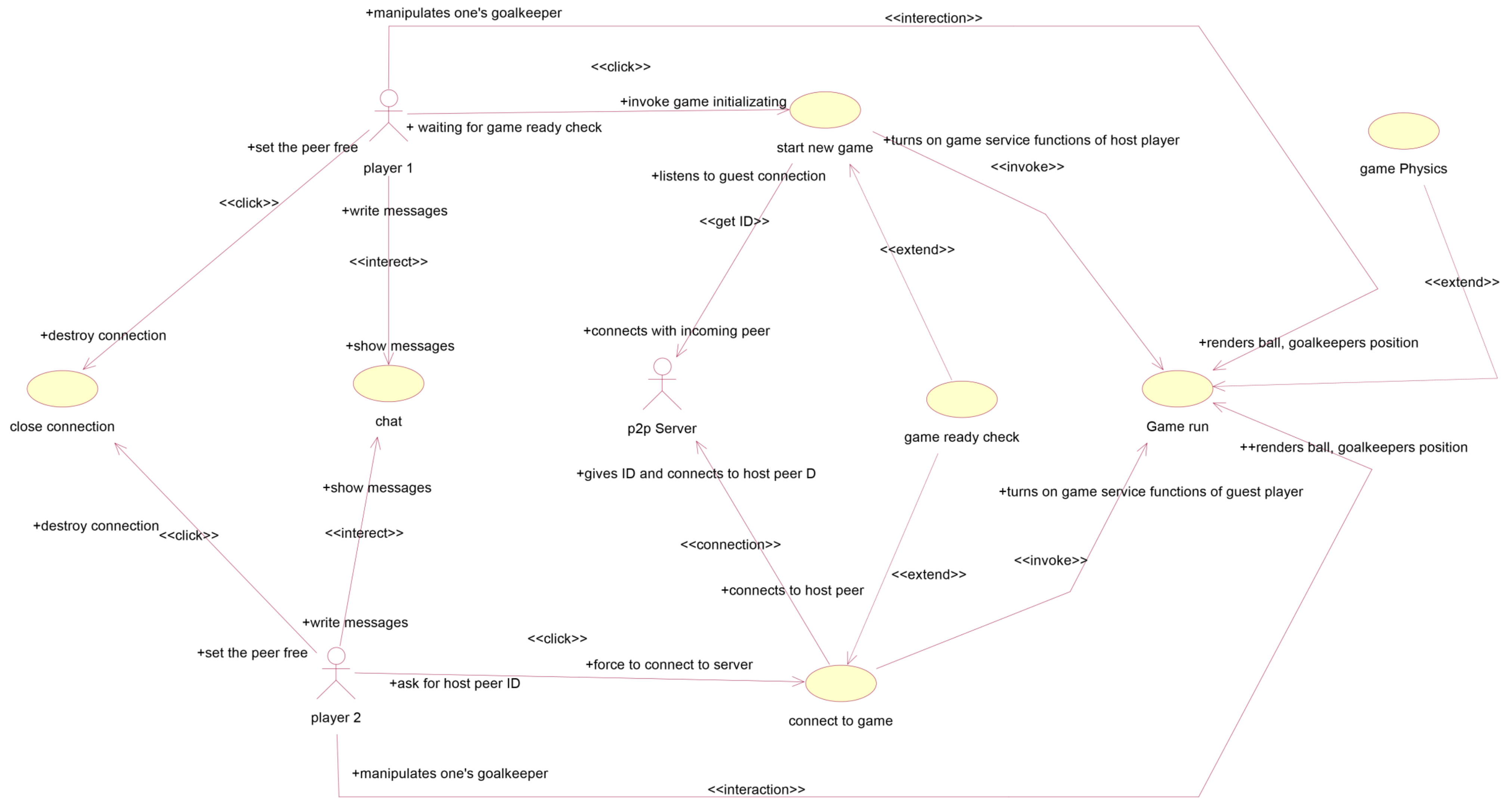
Unsolved: This bug is similar to the first that we have solved (when checkStatus was lost). But in this case the problem is more complicated. This bug appears after repairing after internet connection lost when User1 cannot send data to User2 and User2 can`t receive it. It seems like dataChannel after reconnection doesn’t work properly. User2 is like a host Peer creates the game (User1 only connects to the game) has a Peer.on('connection') method. He also has connection.on('open') method. We tried to handle with both this methods. But the problem is that at start when we establish the first connection these methods works fine. Game starts and goes on. But if you disconnect manually your internet connection than turn it on again you face mentioned problem.

What we do next?

Because it is an beta version (an early version of a program or application that contains most of the major features, but is not yet complete) we also need to work on the code optimization, work on physics of game (more attractive and interesting ball flying directions), we also need to rearrange some logic structures and dive deeper into the webRTC API.

Improved design and responsiveness of the game will be available soon.

Picture – football game use cases



+manipulates one's goalkeeper
<<interection>>
<<click>>
+invoke game initializating
+ waiting for game ready check
start new game
+turns on game service functions of host player
<<invoke>>
game Physics
+set the peer free
player 1
+listens to guest connection
<<click>>
+write messages
<<get ID>>
<<extend>>
<<interect>>
<<extend>>
+destroy connection
+connects with incoming peer
+show messages
+renders ball, goalkeepers position
close connection
chat
p2p Server
game ready check
Game run
++renders ball, goalkeepers position
+gives ID and connects to host peer D
+show messages
+turns on game service functions of guest player
+destroy connection
<<click>>
<<interect>>
<<connection>>
<<invoke>>
<<extend>>
+connects to host peer
+write messages
<<click>>
+set the peer free
+force to connect to server
+ask for host peer ID
player 2
connect to game
+manipulates one's goalkeeper
<<interaction>>