Interactive Digital Systems Hand-in #1

This Hand-in has been written by our group, consisting of:

Name:	Student number:	Student mail:
Dorian Sousa Calvo	74829	dorian@ruc.dk
Jonas Skjoldrup	71333	jdns@ruc.dk
Mathias Larsen	70434	makirk@ruc.dk
Oskar Larsen	71428	oil@ruc.dk

p5.js Prototype

Table of contents:

Project concept

Our IDS project "PVP pong", is intended to be a simple project that puts our recently-acquired skills to use in trying to tackle how we connect two ESP32-devices to the internet, as MQTT.
The two devices, connected to the internet, provide input to a pong-game hosted on a PC. The pong-game part of the product is <u>not</u> our main priority in this hand-in, but rather the establishment of a connection between the two "remotes" (ESP32 devices) and the hosting PC.

Project description

In this segment we'll be describing our project in detail, this includes our code and design of the data-structure of the product.

Design

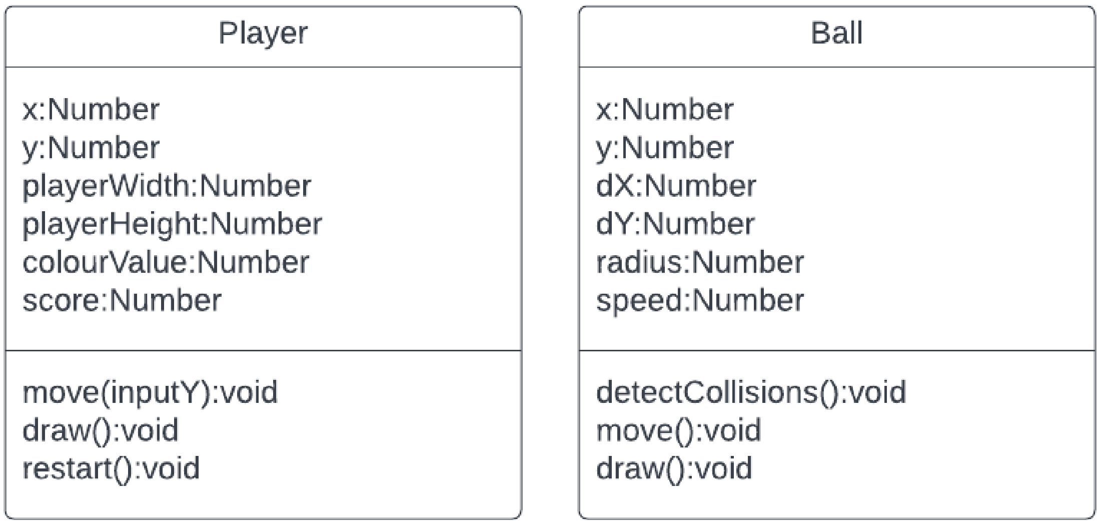
For conceptualizing this project's product, the choice was between many options suggested by lecturers. Since game development and multiplayer is ever growing in relevance this project will look into how you can connect two ESP32 chips to each other and have them both control a player each in a game of pong. This game of pong will be running on a separate machine through p5*js.

ESP32: Wireless controllers

To control the players in the pong game, two simple Arduino programs have been developed and uploaded to two ESP32's. Attached to the ESP32 chip is a joystick which will be the main user control interface. The choice of protocol, being MQTT, brings in some issues in itself, since the broker does not put id's on the signals. This means that the signal sent by the ESP32 itself has to be different from each of the ESP32's. With this, we have the reasoning for the signal being different for the two chips. The signal for player one is (1,-1), while the movement for player two is (2,-2). This signal is then converted on p5*js to actual movement in the game. The code below shows in pseudo code the main loop for the controllers, containing a delay to buffer the inputs sent.
If no delay were to be implemented, the controller would end up sending too many input values. Since MQTT is built on top of TCP the data transfer is guaranteed, meaning that every movement value is sent making it so that sometimes the player moves more than expected. Therefore to avoid that a delay has been implemented making it so that a movement value is only read and sent every 100 milliseconds, or 10 times a second. Since the values the joystick sends are very precise, a deadzone has been implemented, so that values between 3000 and 1000 will not be sent.
Relevant pseudo code:

```
void loop(){
     if (not connected){
          connect()
     }
     if (millis - Lastsignalsend) {
      if (ReadInputFromJoystick() > 3000) {
           sendSignal(-1)
           LastSignalSend = currentTime()
      }
      if(ReadInputFromJoystick() < 1000) {
           sendSignal(1)
           LastSignalSend = currentTime()
      }
    }
}
```

p5*js: PVP Pong



For our game we use two classes: Player and Ball. Ball is a class in case we want to expand and have multiple balls in our game.
Player objects represent the player models in the game and have methods that allow the models to move, to be drawn and restart(), which will reset their y coordinates.
Ball represents the ball in the game.

Setup(): Setup of MQTT in p5*js

Starting the sketch will call the function setup(), besides from creating player and ball objects it will connect the sketch to a broker as a sender and subscribe to two other brokers one for each controller.

```
client = mqtt.connect(
    "wss://public:public@public.cloud.shiftr.io",
    {
      clientId: "ponggame",
    }
  );
  client.on("connect", function () {
    console.log("connected!");
    client.subscribe("/idseplayer1");
    client.subscribe("/idseplayer2");
   });
```

Here we also define how we process messages coming from brokers the sketch subscribed to. For our use we need to change the type of the message from String to Int, by passing the message into a parseInt() function that will return an Int:

```
client.on("message", function (topic, message) {
    number = parseInt(message);
  });
```

Draw(): Sending MQTT messages

draw() will be called repeatedly after setup(), so it's basically a loop.
Starting the sketch sends out a message to the controllers, which will turn on their blue LED to show that a connection has been established:

```
client.publish("ponggame","on");
```

Launching the actual game requires the user to press the Space Key when prompted to:

```
if (keyCode===32) start=true;
```

To move the players we process the receiving values that are deliberately chosen to indicate what player wants to go where.
We use if statements to determine whose player´s input we are working with and passing the value as a parameter in the players move() function will move the player object in the desired direction:

```
if (Math.abs(number)==1){
      player1.move(-number);
    }else if(Math.abs(number)==2){
      player2.move(-number/2);
    }
```

After moving we redraw the players and move on to the ball:

```
ball.move();
    ball.detectCollisions();
    ball.draw();
```

MQTT

The connection, in theory, is reliant on a streaming of player-input to an external client: the game-hosting PC. In more technical terms, we'll be using Message Queuing Telemetry Transport (MQTT).
The transportation of data between the three parties of our system consists of WIFI-connections, in which the two ESP32-devices have the Internet Protocol (IP)-address of the game-hosting PC hard-coded into their programming, allowing them to send data-packets of player-input to the p5*js pong game sketch:

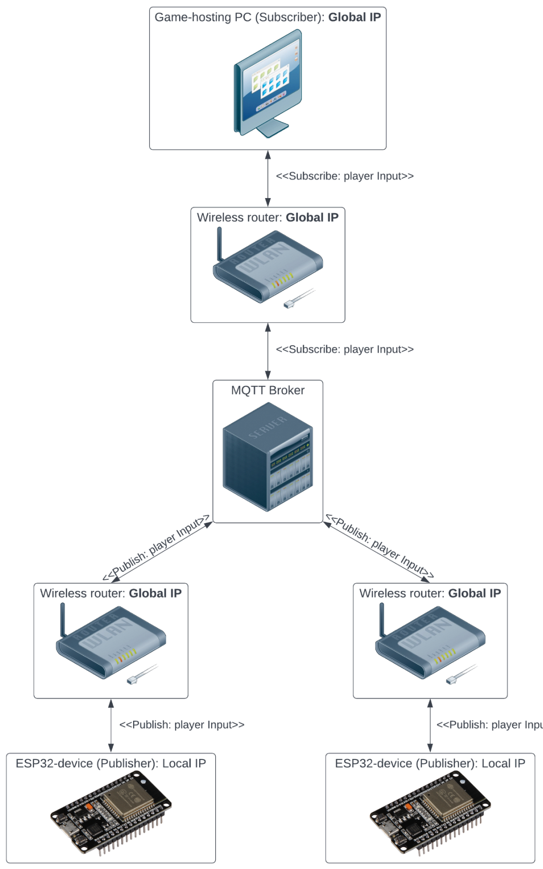

Game-hosting PC (Subscriber): **Global IP**
<<Subscribe: player Input>>
Wireless router: **Global IP**
<<Subscribe: player Input>>
MQTT Broker
<<Publish: player Input>>
<<Publish: player Input>>
Wireless router: **Global IP**
Wireless router: **Global IP**
<<Publish: player Input>>
<<Publish: player Input>>
ESP32-device (Publisher): Local IP
ESP32-device (Publisher): Local IP

Diagram of the three parties involved in our system design: The game-hosting PC, and the two EPS32-devices (acting as wireless controllers).

MQTT is a messaging protocol based on publisher- and subscriber-roles, delegated to each device connected to the internet in the network design. In our project, we're delegating the game-hosting PC to be the subscriber of the two ESP32-devices, as producers. The ESP32-devices send various messages, but the game-hosting PC will be subscribed to the player-input topic that is expected to be received.
We're using MQTT, because it's designed for messaging-connections between remote locations via the internet, also because we're sending small amounts of data.