Pong with Arduino using Node.js and WebSockets

Christian Cardin
Mobile Computing, Services and Security
School of Science
Aalto University
Espoo, Finland
Email: ccardin.sech@gmail.com

Jun Wu
NordSecMob
School of Science
Aalto University
Espoo, Finland
Email:jun.wu@aalto.fi

I. Introduction

This report will introduce a online Ping Pong game which developed based on wireless modules of arduino. The project comprises with three parts: client application, control module consisting of two arduino devices and a web server. The webserver is deployed in cloud service Heroku, control module transmit controlling informations to web server via Zigbee wireless communication protocol and sockets. The game supports multiple "human vs machine" games running at the same time. Audiences can watch arbitrary game with web browser.

II. High Level Architecture

The application relies on three different components: two **Arduino modules** that capture the user's gestures, a **client application** that displays the game graphics and a **remote webserver** that incapsulate the game logic and handles the connections.

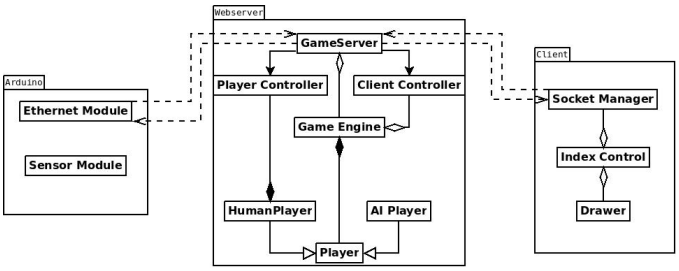

Fig. 1. The high level application architecture.

The principal idea behind this architecture is to build a sort of "game rooms" where the user can either play in a single player mode, choose an opponent and play multiplayer or spectate one of the currently running games as a passive observer. The three parts are logically separated from each other for making them as much reusable and flexible as possible, in such a way that if one part is heavily modified it does not affect the others. In fact the interaction between the components is very simple, based on message exchange. Considered the nature of the system, it is important that the communication between the Arduinos, the Webserver and the Client should be as fast as possible, a real time application. For this reason we chose to implement the communication stream using socket connections using an upcoming technology called **WebSockets**: it allows fast data exchange with a little overhead giving to the end user the sensation to play smoothly and actually in real time.

The communication between components is straightforward: the webserver listens for incoming connections and when a device or a client wants to establish a contact it starts a little handshake, as explained in Fig. 2.

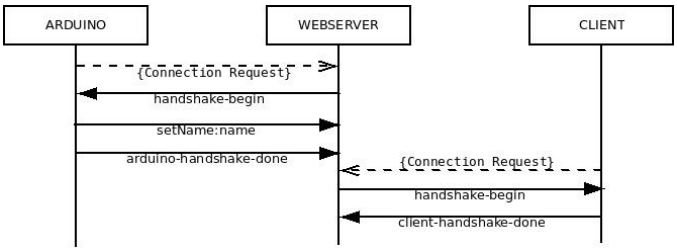

Fig. 2. Handshake and data flow between components.

When a device wants to connect, it sends a message for starting the handshake process. Than, the server replies with a *handshake-begin* indicating that the connection is accepted. Now the device has the possibility to send specific messages to set a series of options in the server, at the moment the only one available is the *setName* message, that tells the server the name of the device. Each device must have a unique name. Finally the device sends a *arduino-handshake-done* for declaring that it has sent all the preliminary information and now it's ready. Received this message, the server registers its name in a list and make it visible to the clients. A similar thing happens with the client, but for the moment there's not additional options, so it just sends back a *client-handshake-done* to confirm its willing to connect.

It's important to notice that the messages can be sent in every moment, maybe one or more requests at the same time, and the server should reply to them in an **asynchronous** way. See the section III for detailed information.

III. Node.js Server

We chose the Node.js framework for several reason:

- **Scalability**: Node.js doesn't use threads to handle the requests, saving computational effort and increasing the number of maximum possible connections.

- **Event driven**: The event-loop is efficient and fast; asynchronous non blocking execution.
- **Ready to go**: With extensions like *Express* and *Socket.io* it's easy to configure it for replying to HTTP requests and instauring socket connections.

It have three main functions: to serve the client application (HTML pages) to users, handle connections and data flow using WebSockets and executing the games. For all the time that the client application and the Arduino are alive the connection is kept open in case something happens: the server is able to make push notification and advise instantly its clients for every modification or event. This could be done because there are two internal lists that map all the open sockets with a unique name, divided into players and spectators. For example, when a new device connects, the server sends a notification to every connected client telling them that some player is now available (and also telling to update their list of available players).

A. Putting all together

The main process creates a socket and listens for incoming connections. After a player connected his Arduino and successfully done the handshake, the server wraps the socket in a class "***PlayerController***" that offers a layer of abstraction on it; then saves this new created object in a dedicated list of "Players". A similar thing is done for those who opens the client application ("***ClientController***" objects that make "Spectators").

For every spectator, the server creates listeners waiting for these possible events: *newGame:playerName*, *spectateGame:gamenumber*, *leave*, *disconnect*. When a *newGame* event is received the server uses the *playerName* field to retrieve the PlayerController object relative to that device and gives it as initial parameter of a new ***HumanPlayer***. Finally, the server instantiates an AIPlayer, that implements an artificial intelligence to play against the human player, and creates an instance of a new ***GameEngine***. After this operation, a progressive number is assigned to the created game, the player is removed from the list of available players and the clients are notified with a new updated list of games and players. There could me more than one game running at the same time thanks to the asynchronous nature of Node.js.

B. Running a game

A game is represented by an instance of GameEngine. This class takes as initial parameters two instances of players (doesn't matter if they are human or AI) and defines an internal list of spectators. It's possible to dynamically add and remove spectators to this list to make them "watch" at the running game, but for now it must have at least one spectator, otherwise it is forced to stop.

The game is organized in three parts: initialization, game-loop and score.

- **Init**: Messages are sent to all Spectators with information about the game initial status and constants (bar dimensions, ball position, score...). Players are advised to start the transmission.
- **Game-Loop**: Every FRAME_RATE seconds the game executes a loop. In this loop the *update()* function of both players is call, that makes the height of the bar to change. In the case of a HumanPlayer, the function reads from the socket connected to the Arduino the last transmitted value for the height (in a value between 0 and 1000) and for the AIPlayer calls a routine that calculates the new position acting something like intelligent. Also *moveBall()* is called, making the ball to move in the field. Then, for every frame a message is sent to all Spectators containing the updated positions to display.
- **Score**: When a player make a point the loop is interrupted. If a player wins the game is terminated, instead it returns again to the Init point.

C. Motion System and Artificial Intelligence

After tests we found the stability of sensor input is not perfect so that it is not easy to control the movements of the bar by human. Hence we only implemented a simple AI which will not make the game too difficult.

The speed of the ball is represent by a float pairs $< x, y >$, x represents horizontal speed and y represents vertical direction. The float number means relative length(according to the side length) the ball moves in one second and the sign before represents the direction. For example, if $x = -0.5$ it means the ball would move half the width from right to left in next second.

To make each set is different, game engine will generate a random speed to the ball at the beginning. The absolutely value ranges in $[0.2, 0.33]$ for x and $[0.0, 0.5]$ for y. The lower bound 0.2 is to make sure the ball would move towards one of the participants in a speed which is not too low.

If the initial speed in horizontal direction is too low, the game would last for a long time and seems boring. AI would also be difficult to beat in this case. Hence we add a rule to increase the pace: the horizontal speed will increase 3% whenever it touch the bar. The collision between bar and ball also produces an effect on vertical speed. Game engine records the speed of both bars and whenever the collision happens, bar's speed will be added to ball's speed. This rule ensure that player can manipulate the ball to move to specific target position.

These are two basic strategies for designing artificial intelligence: first is moving according to ball's current position and speed, second is predicting the final position of the ball and move there. In our implementation we only adopted first one, the opponent would compare the position of ball and its own position, if the opponent is under the ball, it will move up, otherwise move down. The speed of opponent is fixed to 0.3. It is obvious that we can adjust the difficulties by changing the opponent's speed.

IV. Arduino Devices

The control side includes two arduino devices, two thinkerkits , two Xbee modules, one LED and one accelerometer.

A. Sensor Station

The task for sensor station is easy: collect data from accelerometer, format it and send to ethernet station. We read the $inclination$ value from the sensor which calculated by two values from both directions. Then apply a smooth function which calculates the average value of previous 20 data on it. After smooth, we cut it to a range $[-80, 80]$(in this range values are stable) and map it to $[0, 1000]$ as the latest position of player. Then send it to ethernet station via Xbee module. In tests we found that accelerometer sometimes return error value(0), we removed them in the implementation.

B. Ethernet Station

Ethernet station is responsible for maintaining two connection, one is between it and web server, another is the connection to sensor station. In sensor station side, it consistently receive the values from sensor station and update it to a buffer. In another side, first it will perform a simple handshake process to create the connection between it and server. In the process of playing, ethernet station will receive a heartbeat message from web server every two seconds. After that it will response with a heartbeat message. One LED is used to display different states for the connection. For example, blinking means the device preparation process is finished and connection between it and web server is OK, it is available to start the game.

C. Xbee modules

Xbee modules are easy to configure. First obtain X-CTU software from Internet, then configure the parameters of both Xbee modules. The important parameters are channel(set as the same one), PAN-ID(set as the same private network id), DH&DL(destination address) and MY(my address). After that these two arduino devices can communicate by serial.print and serial.read functions.

V. HTML Client

The Client Application is a single page web application, that uses HTML5 attributes and Javascript to display the game and handling the message from and to the server. We made this choice because in this way every device that can run a (decent) browser can enjoy the final product. The advantage is very big: no installation needed, cross platform compatibility, usable even from smartphones. In addition to this, the HTML5 Canvas API are very easy to use and thanks to the CSS we could achieve a graphic adaptable to any screen resolution, automatically and dynamically adjusted. We adopted **jQuery** and **socket.io** libraries to help us in the work.

The frontend part consists of one single *index.html* page and three javascript files. The entry point is *indexcontrol.js*: Binds user actions with functions, deals with the player and game list recovery and update it in the page, and finally binds the events coming from socket to specific actions (visualize the game field, set game name and score, make the canvas to update). It creates two support objects, a **Drawer** and a **Socket Manager**.

The Socket Manager wraps the socket object in a handy class more easy to use. It can bind and unbind functions to events coming from server, and it cares about doing the initial handshake and maintaining an active state.

The Drawer object handles the Canvas space, draws the bars and the ball. Every time that an update comes from the server, the drawer is entitled to change the position of the elements in the Canvas and reflect the last state of the game.

VI. Discussion

Future works can be done in following aspects.

- **Better Sensors**: In tests we found that accelerometer is not a good choice for controlling. The values varies a lot even when the sensor is fixed in a position. Other sensor such as gyroscope sensor should be tested.
- **Complex Artificial Intelligence**: If we own a good control device, the game will become too easy. It is necessary to implement the predicting strategy which mentioned above.
- **Human vs Human**: Current implementation of this pong game only supports ”Man vs Machine” mode. The game engine can be easily extended to ”Human vs Human” situation but a lot of work should be done on interaction parts. Series of mechanisms such as ”Invite Opponent” and ”Randomly Choose Opponent” should be considered. We should also add the option that player can reject game invitations from other players.
- **Verification Mechanism**: The game is lack of secure mechanisms. For example, player can arbitrarily choose a controller and start the game which means you controller may be occupied by others.
- **Multiple Control Options**: Besides accelerometer, we can add more controlling ways such as playing with mouse. In fact, the one who clicks ”start the game” is strange in current implementation. Because after this action there is no difference between him and audiences. We can make it play with mouse and support ”Man vs Man” in future work.
- **Multiple Sensor Stations Connecting to Single Ethernet Station** :Currently, a player comprises with two arduino devices so that each player should have a ethernet station. It is more effective to make multiple sensor stations connect to single ethernet station. To achieve this goal, a new protocol between sensor stations and ethernet stations should be designed.

VII. Conclusion

In this paper we introduce an online Pingpong game. The project combines various techniques ranging from wireless sensor network, cloud service and HTML/javascript technique. Multiple players could play with a simple AI at the same time. Moreover, audience who owns an Internet device can watch the live game everywhere.