HTML5 player-vs-player game server

Tsang Hao Fung

Introduction

As HTML5 technology becomes more mature, complicated games become possible in browsers. With the rising of social media network, “friends” become more important than before, making multiplayer functionality a very important factor for a successful game. However, game designer and developers often find it very hard to add this feature. In this project, we try to provide a one-stop solution.

Problem statement

We want a game server that can support multiplayer gaming, as well as a chat room for players to pair up with each other. We also want a client library that is easy to integrate into existing games.

Only player-vs-player gaming involving 2 peers is concerned. Supporting more players requires more complicated protocol and server. Indeed, most casual games are not very well-suited for more than 2 players.

Technology

The good old Apache+CGI architecture does not work well for “real time web”. Mojolicious is a web framework designed exactly for this new paradigm. In “real time web”, server process is persistent rather than request-spawn-respond-die. In other words, the server process stays alive and handle clients on a persistent webSocket.

Architecture

Our “HTML5 player-vs-player game server in Perl” (abbreviated as PvP server) has 4 services:

- static HTML server
- login / authentication
- chat room
- peer-2-peer gaming

In the current implementation, they are put into one small program. When the number of users grows, we can split them to several servers.

User flow

1. Login
2. connect to the chat channel and chat with other players
3. start a game with a player
4. redirect to the game page
5. connect to the peer channel and start the game

Protocol

It has an initial handshake part for clients to acknowledge each other. Then we use a runtime protocol known as “lockstep” or “ping-pong”. Basically each client has to wait (locked) until it receives a packet from the other side to increment the timer and perform one frame of computation. After that, a client returns the result of the computation during that frame.

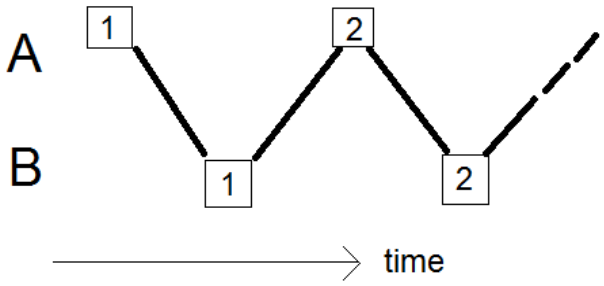

Figure 1. Illustration of a ping-pong protocol

active	passive
1. initiate a handshake 'hi'	1. wait
2. receive 'hi back'	2. receive 'hi' -> 'hi back'
3. handshake success	3. handshake success
4. game start 'time=0'	4. wait
5. receive 'time=0'+gamedata	5. receive 'time=0'
-> 'time=1'+gamedata	-> 'time=0' + gamedata
6. receive 'time=1'+gamedata	6. receive 'time=1'+gamedata
-> 'time=2'+gamedata	-> 'time=1'+gamedata
7. ...	7. receive 'time=2'+gamedata
	-> 'time=2'+gamedata

Figure 2. Data flow of our protocol

Security

Right now the login policy is allow everyone to login, as long as they use a unique username. To keep things simple, cheating is not concerned, and we always trust the clients. The real world is much more dangerous, so this server can only be used in your pet project, for fun only.

Client side

A client library is provided such that developers can setup by only a few lines of code, provided their game is built correctly. In particular, the game has to be deterministic and synchronizable. Here are some general rules:

- random numbers have to be seeded

  https://github.com/tyt2y3/F.LF/blob/master/third_party/random.js
- use a fixed interval game clock

http://project--f.blogspot.hk/2013/04/time-model-and-determinism.html

- all user input has to be buffered
  http://project--f.blogspot.hk/2012/11/keyboard-controller.html