卒業制作

HTTPを扱ったリアルタイムサーバを介してのオンライン対戦タイピングソフト

情報処理学科2年

K198017

若林 宏樹

目次

第１章　はじめに

1-1 作品制作に至った経緯

1-2 目的

第2章　技術要素

2-1 開発環境

2-2 通信形式

HTTP, WebSocket, WebRTC(P2P)

2-3 オンラインゲームにおける快適さ

直接通信型、サーバ集中型、クライアント分散型の処理の差

2-4 マッチングシステム

第3章　作品説明

3-1 作品概要

3-2 作品構成

第４章　制作を終えて

　　4-1 考察

　　4-2 おわりに

第5章　参考URL・文献

第１章　はじめに

1-1 作品制作に至った経緯

ゲーム業界でフルスタックエンジニアとして大成する事を志している身として、オンラインゲームの制作を一連の流れを通して行いたいという想いがあった。その中で、情報処理学科内で行われているタイピング練習やタイピング大会に不便な点や改善出来る点を感じた為、今回タイピングソフトというテーマで開発を行った。

また昨今ではオンラインゲーム制作・運営に当たって、AWSのGameLiftやPhotonのようにバックエンド側に精通していない人間でも扱えるオンラインゲーム制作向けのサービスを利用する事も増えている。ただ、今回は一連の流れを通してサーバ側の技術への知見を深めるという観点から一定量利便性の低い状態のレンタルサーバを扱った制作に踏み切った。

1-2 今回の作品制作の目的

まず私自身の個人の目的として、4月からサーバエンジニアとして働く身として、オンラインゲーム制作全体の流れへの理解を深めるという目的で開発に取り組んだ。

次に、制作における目的としてオンラインゲームをプレイする際のプレイヤーの快適さの追求を設定した。オンラインゲームのプレイにおける快適さとしては、”通信部分のラグ”といった部分がストレスになるという考えを元に快適さの向上を目指した。その一環として使用する通信プロトコルの選定とクライアント-サーバ間での通信方式に重点を置き制作を行った。

第2章　使用技術

2-1 開発環境

【使用デバイス】

OS：Windows10 64bit HomeEdition

プロセッサ：Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz 2.21GHz

実装RAM：8.00GB

【サーバ】

AWS EC2 AmazonLinux

使用言語：PHP7.1

【クライアント】

Unity2019.4.7f1

使用言語：C#

【バージョン管理システム】

SourceTree(内臓Git バージョン 2.20.1/LibGit2Sharp 0.25.4+g06e64a2235.libgit2-4aecb64+ git-lfs v2.6.1)

2-2 通信形式

【通信形式概要】

リアルタイム通信を行うオンラインゲームを制作するに当たってサーバとの通信は必須であると考えた。

クライアントとサーバ間の通信を実現する為にはポートの確保や通信管理を行うプロトコルが必要であり、TCPとUDPの二つに大きく分けられる。簡単に説明すると、TCPはコネクションを確立して行う通信方式でUDPはコネクションを確立せずに送信側から一方的に通信を行う通信方式である。この仕様からTCPでは確実性は高いものの効率や速度で一歩おとり、UDPでは効率や速度では優位なものの確実性は相当に落ちてしまうというそれぞれ一長一短の物となっている。

また、通信プロトコルの中でもTCPではWebSocketや今回したHTTPが、UDPにはWebRTC(P2P)等があるというように、偏にオンライン対戦を実現すると言っても様々な通信規約がある。

ここでは何故HTTPを選択したのかを、他の通信規約の解説も交えて説明していく。

【各通信規約についての説明】

・HTTP

　HTTP(Hypertext Transfer Protocol)はリクエスト-レスポンス型の通信プロトコルであり、クライアント側がサーバ側にリクエストメッセージを送信してサーバ側がそれに対してレスポンスを返すといったもの。サーバとクライアントの通信において最も基本的な通信プロトコルと言える。

　クライアントとサーバのコネクションは毎回確立する必要があり、クライアント側からのリクエストの際に確立を行う。この特徴から基本的にはサーバ側から能動的にクライアント側にアクションを起こすことは出来なくなっている。

　また通信の度にクライアントとサーバのコネクションを確立してからリクエストを送信しレスポンスを受け取るという流れを汲む為、データを一方的に送信するという形のUDPより通信の信頼性は高いが少なからずデータのやり取りにかかる時間は増加している。(図2-1参照)

　今回は対戦相手の行動がプレイヤー側に影響を及ぼさないシステムである為に完全同期の必要性が無く、完全同期とすると対戦相手の通信環境によりストレスの要因になるという考えから非同期型であり最も基本的なプロトコルであるHTTPを選択した。

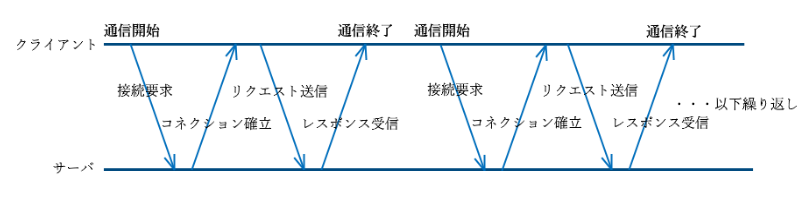


図2-1 HTTPの通信の流れのイメージ図

・WebSocket

　WebSocketプロトコルは双方向通信を実現させるために作られた通信プロトコル。最初はHTML5の仕様として策定されていたが、現在は単独のプロトコルとして存在している。

　大雑把な流れとしては初めにHTTPのようにしてコネクションを確立し、以降TCP/IPのように振る舞うといったもの。

　この方式では一度コネクションを確立した後は、サーバとクライアントの双方から通信を行う事が可能になっている。これが双方向通信と呼ばれる所以である。

　また、一度コネクションを確立した後はそのコネクション上で通信を行う他、ヘッダのサイズが小さくなっており通信量の削減にもかっている。

　このように言うと良い事ずくめのようだが、前述のメリットはコネクションを同時に一つしか持てないサーバでは複数の通信用のサーバを確立する必要があり相性が悪い物だとデメリットと表裏一体となる可能性もある。

　今後HTTPに統合される可能性があるという話もあり、伝統的な通信規約であり私自身が使っていく事が増えるであろうHTTPを今回は選択した。

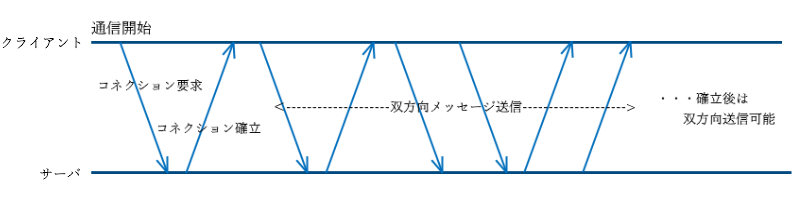


図2-2 WebSocket通信の流れのイメージ図

・WebRTC(P2P)

P2Pはそもそもクライアント-サーバ方式ではない通信規約である。ネットワークに接続されたコンピューター同士が対等な立場、機能で直接通信を行う物である。

デメリットとしては、データを共有する全てのコンピューター同士が通信をするという仕様上、接続するコンピューターが増加するほどに通信経路が複雑化していってしまうという物がある。例えば2台だと通信経路は1つ、3台だと通信経路は3つだが、4台になると6つ5台になると11つと指数関数的に複雑になっていく事が分かる。

他にもネットワークの外部からプライベートIPアドレスにはアクセス出来ない為にNAT越えが必要になる等、工程が複雑化してしまう事が挙げられる。

メリットとしては、間に介する物が純粋に少ない(お互いのコンピューターとネットワークのみ)為、少人数での通信であれば最小限のラグで実装出来るという物がある。またクライアント-サーバ方式では大人数のアクセスによってサーバ及び回線に負荷がかかる事もあるが、P2P方式では通信が複雑になるという部分に目を瞑れば接続数が膨大になっても特定の機器にアクセス集中が起きにくくなっている。

今回の作品では1vs1の対戦方式であり、観戦者も含めても大人数での接続という事にはならないのでこの方式での実装も考えた。しかし、今回のリアルタイムサーバを扱うというテーマから逸脱しており、また完全同期という特徴から対戦時のストレスに繋がるという考えからこの方式での実装は見送った。

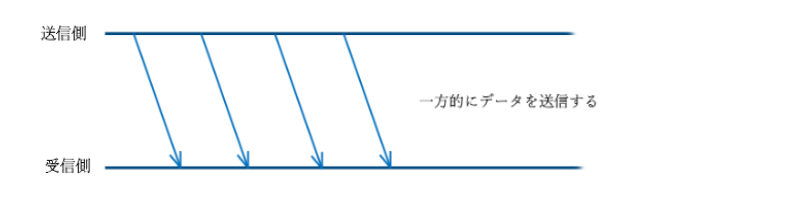


図2-3 WebRTCの流れのイメージ図

2-3 オンラインゲームにおける快適さ

【オンラインゲームにおける快適さに対しての考え】

　私はオンラインゲームにおける快適さは所謂『重さ』に依存していると考えた。

　この『重さ』がゲームのストレスに影響して来る場面は起動時、シーン切り替え時、ゲーム中といった物が主だったものだろう。ここでは特にゲーム中のリアルタイム通信に関する物に着目していこうと思う。

　『重さ』は通信、計算、描画等が影響して来るのだが、それをサーバとクライアントにどの程度分散させるのかで調整が可能ではないかと予測し、処理の分散について注目した。

　また今回のシステムでは対戦相手の行動がプレイヤーに及ぼす影響が心理的な部分のみの為、対戦者同士の同期よりもプレイヤーの操作に対するレスポンスの滑らかさが対戦中に『重さ』をプレイヤーに感じさせない為に重要だと考え、『プレイヤー動作の処理＞対戦相手の描画の同期』という形で優先度を設定した。

【処理分散の方式】

・クライアント分散型：非同期型

　今回採用したのはこの形式。

　プレイヤー自身の描画はクライアント側に委ねて、モンスターの動作やクライアント間の同期などはサーバ側で行うという物。どこまでをクライアント側に委ねるかはゲームにも寄るが、通信相手やゲーム世界に影響が大きい部分はサーバ側で処理を行い、そうでない部分はクライアント側で処理を行う事が多い。

　MMOやMOのRPGやアクションゲームで採用されている事が多い。有名どころのゲームだとPSO2はこの形式を取っているとの事。プレイヤーのキャラクターが移動した後に突然引き戻される、プレイヤーキャラクターは操作出来ても周囲のキャラクターが動かなくなるといった動作が見られるゲームはこの形式を取っていると予測出来る。

　先述したように今回のシステムではプレイヤー同士はタイピング速度を競うのみでお互いに影響を与え合う事は無いので、完全同期を行う必要はないと考えた。また対戦時にプレイヤーの受けるストレスという観点から、タイピングゲームという分野において自分がタイピングを行っているのに判定が行われないという物が最たるものであろうと考え、プレイヤーの動作と対戦相手データの同期を分散出来るこの形式を採用した。

・サーバ集中型：非同期型

サーバ側で全ての処理を行うのでゲーム世界の動作自体は同期しているが、通信遅延などの影響でクライアント側に描画される映像には差が出る事があるという物。プレイヤーの動作はプレイヤーとサーバ間の通信環境に依存する事、他のプレイヤーの通信環境にプレイヤー自身の動作自体には影響がない事、当たり判定等は全てサーバ側で同期させられる事といった点からFPSゲームなどで多く採用されている。

クライアント側で描画される映像に差異が出るとなるとFPSには向いていないように感じるかもしれないが、クライアント側の描画に僅かな遅延を入れる、サーバ側の判定を遡って行うなどの工夫によって成り立てている。他にもこの方式では帯域が渋滞する問題が出る事が多いので通信の頻度を下げる他、それで問題があるものについてはデータを圧縮する等の工夫が凝らされている。

今回はサーバが資金の関係上AWSの無料枠という事でどこまでの負荷に耐えうるかの試験を行う事や、判定をサーバ側に委ねるので自分自身との勝負でもあるタイピングゲームという分野において適していないという考えからこの形式での実装を見送った。

・直接通信型：完全同期型

　これは前述したP2Pの事。互いの処理を互いに行い直接行うので所謂ラグが少なくなっている。正確に言うと、ラグに影響する要素の数が少なくなっている。ただし、ユーザー同士では互いに待ち合わせする必要があるのでレイテンシの高いユーザーに足を引っ張られるという問題は発生する。

　また、P2Pの説明の項目でも述べたがサーバにアクセスが集中するという問題を回避出来るので、同一時間帯に多人数のユーザーのアクセスが見込まれるようなシステムでのアクセス集中の回避の目的でも利用される。

　その為、この方式はマッチングやデータの管理はサーバを介して行い、ゲーム中はリアルタイムサーバ(ゲーム中の処理を行うサーバ)を介さずに直接通信・対戦を行うという物が多い。

　この形式にはキー入力同期方式とコマンド入力同期方式の二つの方式が存在しており、簡単に説明するとデータ送信のタイミングが異なっている。キー入力同期方式では主に1フレームのラグも許せない格闘ゲーム等で採用されている事が多く、コマンド入力同期方式ではターン制のゲームに採用されている事が多い。

私自身も採用を検討したがP2Pの項目で前述したような理由で今回の採用は見送った。

第3章　作品説明

3-1 作品概要

【作品名】

対戦型タイピングソフト『vsTyping』

【作品説明】

タイピングのプレイ形式は、ソロモードとマルチモードを用意。

ソロモードはクライアントのみで完結させており、オフライン時でも遊べるようになっている。ただし、オフライン時の記録はデータベースに反映されない。

マルチモードでは、HTTP通信でサーバを経由してプレイヤーデータのやり取りをリアルタイムで行う事でオンライン対戦を行う。

マルチでのマッチングの形式は『フレンド戦』と『ランダム戦』を用意した。フレンド戦ではホストが部屋を立て、参加者が部屋番号を入力することで入室する。ランダム戦では完全ランダムでマッチングを行い、対戦を始める事が出来る。

第4章　制作を終えて

第5章　参考文献・URL

5-1 参考文献

5-2 参考URL

【Unity ~C#関連~】

・C# ゲームプログラミングはホントにメモリに無頓着で良いの？

<https://www.slideshare.net/KMC_JP/c-91154309>

・Unityでユーザーデータをローカルにファイルとして保存する-Qiita

<https://qiita.com/tricrow/items/ff51bce01ae2139a6fcd>

〇C#でファイルの読み込みと書き込み

<https://gametukurikata.com/csharp/readwritefile>

【Unity ~HTTP接続関連~】

〇UnityでHTTPに接続する-Qiita

<https://qiita.com/ponchan/items/65aeb43e8fea8da0bcac>

・UnityでHTTP通信してみる-Qiita

<https://qiita.com/shun-shun123/items/10c7711b129f8d2b7559>

・UnityWebRequestで簡単HTTP(POST)通信

<https://beyondjapan.com/blog/2020/05/unitywebrequest/>

・UnityからサーバへHTTPリクエストを送り、レスポンスを受け取る

<http://matudozer.blog.fc2.com/blog-entry-31.html>

・初心者が送る UnityでAPI通信講座-Qiita

<https://qiita.com/pchan52/items/feca16ea98289ec31c65>

・古来よりUnity非同期を実現していたコルーチンとは何者か？-Qiita

<https://qiita.com/4_mio_11/items/3030f9a39f59cd68f705>

・Unityのコルーチンの使い方をまとめてみた

<https://developers.wonderpla.net/entry/2014/09/02/205829>

【AWS関連】

〇AWS公式チュートリアル-LAMP環境構築

<https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/ec2-lamp-amazon-linux-2.html>

・AWSでのsu(スーパーユーザー)

<https://www.kannon.link/fuku/index.php/2016/09/19/02/>

【PHP関連】

〇ファイルの書き込み

<https://www.php.net/manual/ja/function.file-put-contents.php>

〇PHPでJSONを受け取るには

<https://qiita.com/hidepy/items/42220523cb2b3eb2c451>

・PHPでPUTの受け取り

<https://www.tmp1024.com/articles/php-put-get-data>

【通信関連】

・究極のゲーム用通信プロトコル”WebRTC”

<https://www.slideshare.net/rotsuya/webrtc-60167675>

【オンラインゲームシステム関連】

〇[CEDEC 2010]ネットゲームの裏で何が起こっているのか-4Gamer.net

<https://www.4gamer.net/games/105/G010549/20100905002/>

〇オンラインゲームの仕組みと工夫

<https://www.slideshare.net/imaifactory/ss-48388661>

・リアルタイム対戦ゲームを支えるリアルタイムサーバの負荷試験のリアル

<https://www.slideshare.net/RuofanYe1/ss-126078760>

【マッチングシステム関連】

・MONOBIT マッチングアルゴリズム

<http://www.monobitengine.com/document/main051.htm>