بسم اللّه الرّحمن الرّحیم

آموزش ساخت بازی چند نفره

مفاهیم پایه

مجتبی قاسم زاده تهرانی

فهرست

معرّفی

ساخت بازی‌های چند نفره (مولتی پلیر) کار سختی است، البتّه کانستراکت ۲ بسیاری از پیچیدگی‌های این کار را برای شما آسان کرده است. برای این‌که بتوانید به نحو احسن از قابلیّت چند نفره‌ی کانستراکت ۲ استفاده کنید باید با نحوه‌ی کار بازی‌های آنلاین، مشکلاتی که در این زمینه پیش می‌آید و نحوه‌ی حلّ آن‌ها آشنا شوید.

اگر از چیزهایی که در این آموزش ذکر می‌کنیم استفاده‌ی اشتباه شود، بازی دچار باگ‌ها و مشکلات عجیب و غریبی می‌شود. با این که شاید خیلی دوست داشته باشید هر چه سریع‌تر بازی چند نفره بسازید، ما به شدّت توصیه می‌کنیم قبل از شروع کار هر چهار آموزش ساخت بازی چند نفره را مطالعه کنید. چون که ساخت بازی چند نفره یک ویژگی پیشرفته است، ممکن است برای تازه‌کاران کمی مشکل باشد.

اتّصال

موتور چندنفره‌ی کانستراکت ۲ بر پایه‌ی **WebRTC DataChannels** ساخته شده است. در کانستراکت ۲ بازی‌کن‌ها به جای این‌که از طریق یک سرور به هم وصل شوند، به طور مستقیم به یکدیگر متّصل می‌شوند.

در این حالت اوّلین کسی که به بازی ملحق می‌شود هاست[۱] (Host) نام می‌گیرد. بقیّه‌ی کسانی که به بازی ملحق می‌شوند فقط به هاست متّصل می‌شوند. بازی‌کن‌هایی که هاست نباشند پیر (Peer) نام می‌گیرند. هاست به عنوان سرور بازی عمل می‌کند. تفاوت اصلی بین هاست و سرورهای مرکزی این است که هاست می‌تواند یکی از بازی‌کن‌ها باشد در حالی که سرور همیشه آنلاین توسّط سازنده‌ی بازی راه اندازی شده است. درضمن خودِ هاست یک شرکت کننده‌ی فعّال در بازی است، ولی سرور خود جزو بازی نیست و فقط بازی را برای بازی‌کن‌ها اجرا می‌کند.

[۱] هاست به معنی میزبان می‌باشد.

البته باز هم می‌توانید بازی‌تان را با سرور مرکزی هم بسازید، به این صورت که بازی‌تان را طوری طرّاحی کنید که هاست یکی از شرکت کننده‌ها در بازی نباشد، و آن را روی یک سرور اجرا کنید. امّا اگر بخواهید این کار را انجام دهید مجبورید هزینه‌ای را نیز برای سرور پرداخت کنید. امّا اتّصال همتا به همتا (peer to peer) خیلی ارزان‌تر برایتان در می‌آید.

یک پیر برای این‌که بتواند به هاست وصل شود باید بداند که هاست در کجا قرار دارد و چگونه می‌تواند این کار را انجام دهد. برای این کار باید آدرس IP او را پیدا کند.

سال‌هاست که استفاده از نسخه‌ی ۴ آیپی (IPv4) در اینترنت دوام یافته است. این باعث می‌شود که چندین کاربر پشت یک آیپی مخفی بمانند. این کار به وسیله‌ی برگردان نشانی شبکه (NAT)[۲] انجام می‌شود. مثلاً در خانه یا محلّ کار، شما یک مودم اینترنت دارید که تعدادی رایانه یا موبایل و... برای اینترنت به آن وصل شده اند، در این حالت گونه‌ای از NAT باعث می‌شود آدرس آیپی تمام آن‌ها با هم یکی شود، در حالی که آن‌ها از پورت‌های مختلف به آن مودم وصل شده‌اند. NAT انواع دیگری نیز دارد، بعضی از انواع NAT باعث یکی شدن آیپی‌های یک منطقه، ارائه دهندگان خدمات اینترنتی (ISPها)، و شبکه‌های تلفن همراه می‌شوند. متأسفانه این یعنی ممکن است در بعضی از موارد امکان اتّصال نباشد، مخصوصاً اگر هاست و پیر هر دو به وسیله‌ی NAT پشت یک آیپی مخفی شده باشند.

مثلاً یک پیر را در نظر بگیرید که می‌خواهد به هاست متّصل شود و هاست پشت یک NAT محدود مثل تصویر بالا باشد. در این صورت به خاطر یکی بودن نشانی آیپی این سه پورت، پیر نمی‌داند که باید به Port A وصل شود یا Port B یا... . این باعث می‌شود که در بعضی از موارد دو بازی‌کن نتوانند به یکدیگر متّصل شوند، و در این موارد باید نقش هاست را بازی‌کن دیگری انجام دهد.

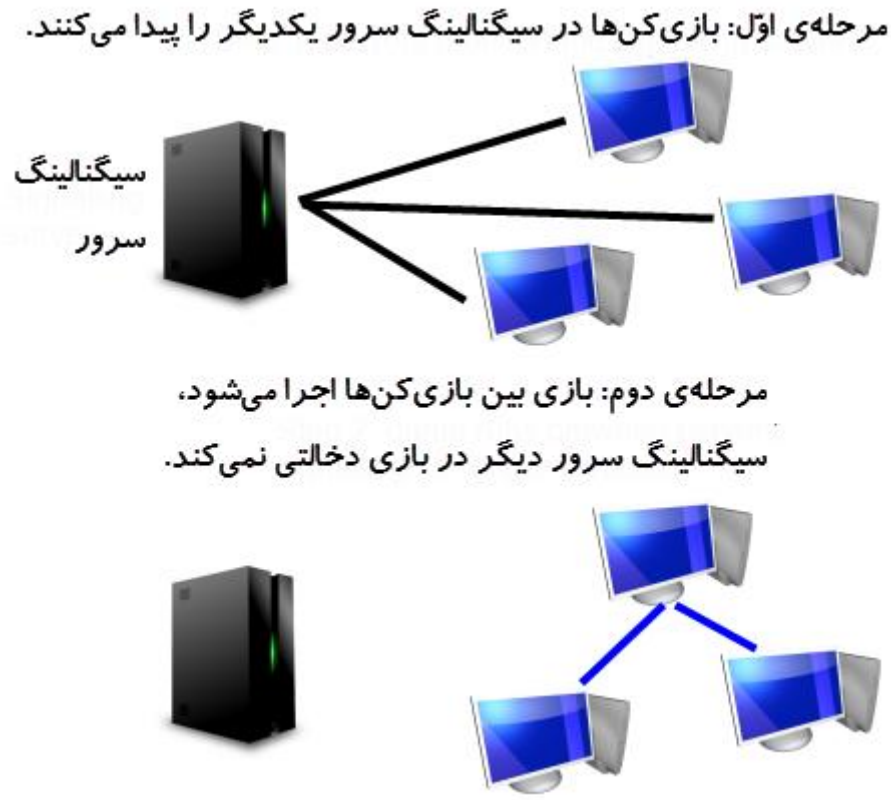
[۲] NAT: Network Address Translation

نسخه‌ی ۶ آیپی (IPv6) با تعداد نشانی‌های بسیار بیشتر از IPv4 به تدریج اینترنت را می‌پوشاند. در این صورت هر دستگاه متّصل به اینترنت یک آیپی منحصر به فرد خواهد داشت و دیگر نیازی به NAT وجود ندارد، و در نتیجه مشکلی در اتّصال نخواهیم داشت.

سیگنال دهی

همان طور که گفته شد یک پیر برای این‌که بتواند به هاست وصل شود باید آیپی آن را پیدا کند و تشخیص دهد که چگونه می‌تواند به آن وصل شود. از طرفی هاست می‌تواند هرکسی، و در هر جایی روی این کره‌ی خاکی باشد، در نتیجه فهمیدن این‌که پیر به کجا باید وصل شود تقریباً غیر ممکن است. بنابراین برای این‌که بازی‌کن‌ها بتوانند همدیگر را پیدا کنند، یک سرور مرکزی به نام سیگنالینگ سرور (signaling server) ایجاد شده است.

سیگنالینگ سرور هاست اصلی بازی‌ها نیست: اوّلین بازی‌کنی که وارد اتاق بازی می‌شود نقش هاست بازی را دارد و داده‌های بازی را به پیرها می‌رساند. سیگنالینگ سرور فقط یک سرور مرکزی است که بازی‌کن‌ها برای پیدا کردن همدیگر وارد آن می‌شوند. وقتی یک پیر می‌خواهد به هاست ملحق شود، سیگنالینگ سرور اطّلاعاتی مثل نشانی‌های آیپی و جزئیّات اتّصال را دریافت و به دو طرف ارسال می‌کند. اوّلین باری که پیر بتواند با موفّقیّت به هاست متّصل شود، سیگنالینگ سرور دیگر در اتّصال دخالت نمی‌کند و دو طرف به صورت مستقیم با هم ارتباط برقرار می‌کنند.



اسکیرا در آدرس wss://multiplayer.scirra.com یک سیگنالینگ سرور عمومی را میزبانی می‌کند. (wss پروتکلی برای اتّصال ایمن WebSocket است، چون داده‌های سیگنالینگ از طریق WebSocketها ارسال می‌شوند. خود بازی‌ها با استفاده از WebRTC کار می‌کنند.)

اصطلاحات مربوط به سیگنالینگ

سیگنالینگ سرور نباید بازی‌کن‌های یک بازی را به بازی دیگری که هیچ ربطی به آن ندارد وصل کند، و باید به بازی‌کنان اجازه دهد که در گروه‌های مختلف یک بازی را به صورت همزمان بتوانند بازی کنند، برای این کار بازی‌کن‌ها به یک *اتاق* (room) در یک *نمونه‌ی* (instance) بازی ملحق می‌شوند.

سیگنالینگ سرور ابتدا بازی‌کن‌های هر بازی را از بازی دیگر جدا می‌کند. بازی‌تان باید یک نام منحصر به فرد جهانی در سرور داشته باشد تا هر کسی در بازی شما دقیقاً به همان بازی در سیگنالینگ سرور متّصل شود. این کار باعث می‌شود که بازی‌کن‌ها اشتباهی به بازی‌های دیگر نامربوطی که کلاً عملکردشان با بازی شما فرق می‌کند وصل نشوند. برای هر بازی که می‌سازید باید از یک اسم در سیگنالینگ سرور استفاده کنید. برای این‌که اسمی که انتخاب می‌کنید منحصر به فرد باشد در اسم بازی‌تان نام خودتان یا شرکتتان را هم بنویسید. مثلاً اگر اسم بازی‌تان Asteroids است، به جای این‌که در سیگنالینگ سرور اسم بازی‌تان را "Asteroids" خالی بنویسید، بنویسید "MyStudio-Asteroids" یا "JohnSmith-Asteroids" (مثلاً JohnSmith اسم خودتان و MyStudio اسم شرکتتان است). در ضمن این اسمی که انتخاب می‌کنید هیچ جا نمایش داده نمی‌شود و فقط در سیگنالینگ سرور به کار می‌رود، بنابراین به خاطر منحصر به فرد شدن می‌توانید هر جزئیّات دیگری را هم که خواستید به این اسم اضافه کنید و لازم نیست نگران باشید.

بعد سیگنالینگ سرور بازی‌کن‌های نمونه‌های مختلف یک بازی را از هم جدا می‌کند. مثلاً شما بازی‌تان را در یک نمونه‌ی پایدار منتشر کرده‌اید و می‌خواهید بازی‌کن‌ها آن را بازی کنند، و یک نمونه‌ی آزمایشی نیز از بازی‌تان ایجاد کردید تا کسانی که قرار است بازی را تست کنند از آن استفاده کنند و همچنین نمونه‌های دیگری نیز از بازی‌تان ساخته‌اید. این باعث می‌شود که تست‌کنندگان نسخه‌ی آزمایشی به اشتباه به بازی‌کن‌های نسخه‌ی پایدار وصل نشوند. چون شاید نسخه‌ی آزمایشی مکانیزم و خصوصیّاتی متفاوت از نسخه‌ی پایدار داشته باشد. حتّی نسخه‌های بازی هم می‌تواند فرق کند، مثلاً بازی‌کن‌های نسخه‌های v1.0، v1.1، v1.2 و... اشتباهی به یکدیگر وصل نمی‌شوند. حتّی می‌توانید نمونه‌هایی برپایه‌ی مکان زندگی بازی‌کن‌ها داشته باشید، مثلاً بازی‌کنان اروپا، آمریکای شمالی و آسیا هر کدام فقط به بازی‌کن‌های واقع در منطقه‌ی زندگی خودش بتواند وصل شود. این دو می‌توانند با هم ترکیب شوند. یعنی اروپایی‌های نسخه‌ی ۱٫۱ به اروپایی‌های نسخه‌ی ۱٫۲ وصل نمی‌شوند و...

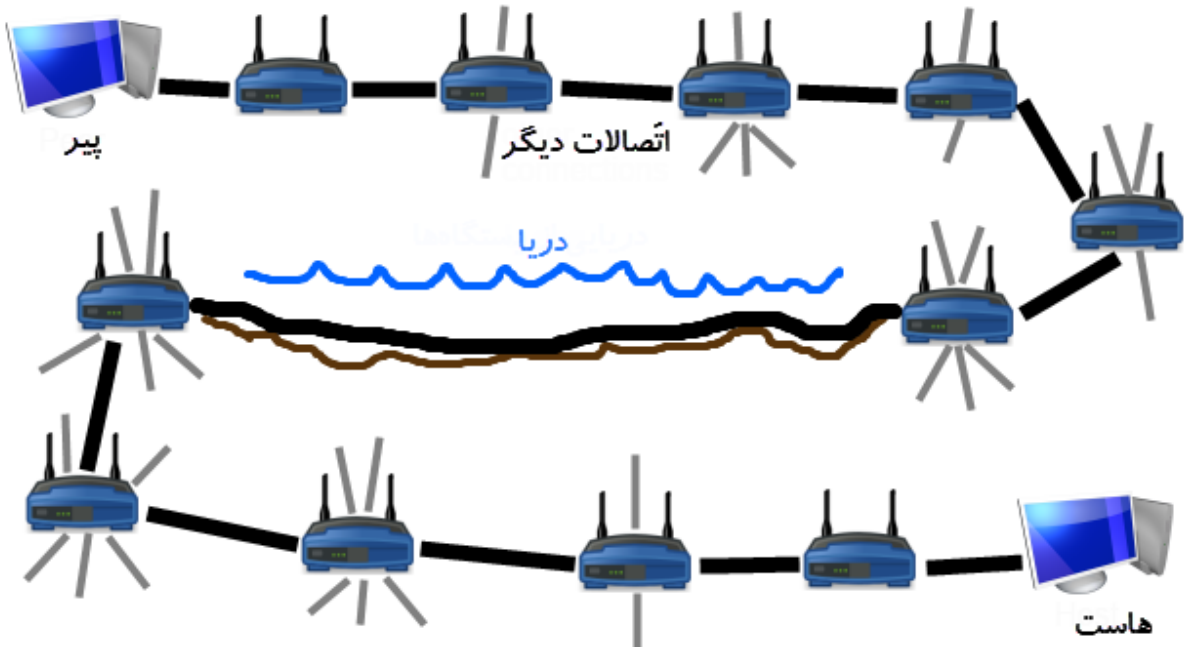
در انتها بازی‌کن‌ها می‌توانند به یک اتاق ملحق شوند. یک اتاق نماینده‌ی گروهی از بازی‌کنان است که دارند با هم بازی می‌کنند. بازی‌کن‌ها نمی‌توانند سایر بازی‌کنانی را که در آن اتاق نیستند مشاهده کنند. اوّلین بازی‌کنی که وارد اتاق شود هاست می‌شود، بعد سیگنالینگ سرور هاست را از ورود هر بازی‌کن دیگری به آن اتاق مطّلع می‌کند. بعد آن‌ها به هم متّصل می‌شوند و شروع به بازی کردن می‌کنند.

در شیء Multiplayer کانستراکت ۲، همه‌ی اکشن‌ها، کندیشن‌ها واکسپرشن‌هایی که به سیگنالینگ سرور ربط داشته باشند در یک گروه به اسم Signalling قرار می‌گیرند. بقیّه‌ی آن‌ها به خود بازی‌ها مربوط می‌شوند.

هرج و مرج اینترنت

خوب. همان طور که گفتیم بعد از این‌که سیگنالینگ سرور نقش خود را ایفا کند در بازی دخالتی نخواهد کرد و نقش هاست بازی بر عهده‌ی یکی از بازی‌کنان می‌افتد. پیرها فقط به هاست اطّلاعات می‌فرستند ولی هاست می‌تواند به هرکسی اطّلاعات بفرستد. یعنی اگر مثلاً دو تا پیر بخواهند مستقیماً با هم ارتباط داشته باشند باید از طریق هاست اطّلاعات را به هم بفرستند.

اگر قرار بود بازی چندنفره فقط در یک شبکه‌ی محلّی (LAN) مثل شبکه‌های خانگی یا دفتر کار اجرا شود کار ساخت آن خیلی راحت‌تر بود. امّا اینترنت پدیده‌ای بسیار پیچیده‌تر است. بین دو بازی‌کن اینترنتی می‌تواند حدود ۱۰ تا ۲۰ دستگاه شبکه (مثل مودم و روتر) وجود داشته باشد. دستگاه‌هایی که در طول این مسیر وجود دارند برای هر جفت از بازی‌کنان متفاوت است. برای هر پیام ساده‌ای که فرستاده می‌شود، هر دستگاه واقع در مسیر باید یک بسته از اطّلاعات را دریافت کند، بخواند، بفهمد که چیست و به کجا باید فرستاده شود، و سپس آن را مجدّداً به دستگاه بعدی بفرستد. حالا اگر شما به بازی‌کنی متّصل شوید که آن طرف آب زندگی می‌کند، احتمال آن خیلی زیاد است که اطّلاعات از طریق کابل‌های زیر آبی که معمولاً ترافیک بسیار زیادی دارند منتقل شود.



هر دستگاه واقع در مسیر به طور عجیبی بسته‌های دریافتی را در طیّ فقط چند هزارم ثانیه[۳] به دستگاه بعدی منتقل می‌کنند. امّا به خاطر وجود تعداد زیادی دستگاه در مسیر، این زمان افزایش می‌یابد. اگر حتّی یکی از این دستگاه‌ها در طول مسیر دیر بارگیری شود باعث می‌شود زمانی که طول می‌کشد تا اطّلاعات ارسال شوند افزایش یابد. این دستگاه‌ها، مثل هر کامپیوتری ممکن است کرش کنند، با کمبود حافظه مواجه شوند، برای نگهداری خاموش‌شان کنند، برق‌شان برود، در ارائه‌ی خدمات دچار باگ شوند و غیره. به خاطر مهندسی پیشرفته‌ای که در این زمینه انجام شده است، در چنین شرایطی روترها بسته‌های اطّلاعات را دوباره می‌فرستند و دستگاه مشکل دار را دور می‌زنند و یا حتّی آن‌ها را از مسیرهای دیگری به مقصد می‌رسانند. همه‌ی این کارها می‌تواند باعث ایجاد تغییرات موقّتی یا دائمی زمان‌بری در مسیر انتقال اطّلاعات شود.

اگر یک بازی‌کن از لندن در انگلستان بخواهد با یک بازی‌کن از سیدنی در استرالیا بازی کند، اطّلاعات بازی باید حدود ۱۷۰۰۰ کیلومتر راه را طی کنند. حالا طبق قوانین فیزیکی اگر در نظر بگیریم اطّلاعات در یک خطّ مستقیم و با سرعت نور ارسال می‌شوند، حدود ۵۷ میلی‌ثانیه طول می‌کشد تا اطّلاعات منتقل شوند. از نظر فیزیکی امکان ارسال اطّلاعات از این سریع‌تر غیر ممکن است. درضمن کابل‌های این مسیر کاملاً در یک خطّ مستقیم قرار ندارند، و در این مسافت بسیار طولانی طبیعتاً دستگاه‌های روتر بسیار بیشتری وجود دارند و زمان خیلی بیشتری صرف می‌شود.

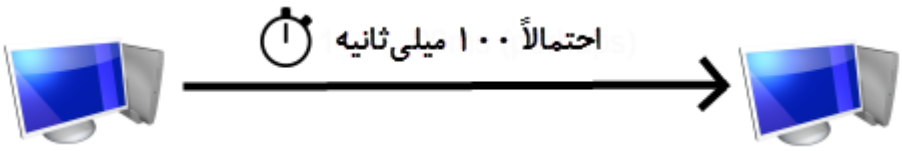
[۳] از این پس به جای عبارت «هزارم ثانیه» از «میلی‌ثانیه» یا همان ms استفاده می‌کنیم.

کیفیّت اتّصال

برای این‌که کیفیّت اتّصال در اینترنت تعیین شود، اندازه‌گیری‌هایی انجام می‌شود. این اندازه‌گیری‌ها در تعیین کیفیّت گیم‌پلی هم مؤثّرند. این اندازه‌گیری‌ها شامل این موارد می‌شود: latency، packet delay variance (PDV)، packet loss و پهنای باند(bandwidth).

Latency

به خاطر دستگاه‌های روتر و قوانین فیزیکی، هر بسته‌ی اطّلاعاتی کمی طول می‌کشد تا به مقصد برسد. به مدّت زمانی که این بسته در راه است Latency و گاهی اوقات زمان پینگ (ping time) گفته می‌شود. این زمان در بین دو بازی‌کن از هر جای جهان که باشند معمولاً بین ۲۰ تا ۲۰۰ میلی‌ثانیه است، البته گاهی اوقات گزارش شده است که Latency از این هم بیشتر بوده است. در حالت عادی نرخ فریم بازی‌ها ۶۰ فریم بر ثانیه است، یعنی هر فریم حدود ۱۶ میلی‌ثانیه نمایش داده می‌شود. حالا اگر Latency بازی ما ۱۰۰ میلی‌ثانیه باشد، یک پیغام که می‌خواهد از یکی از بازی‌کنان به دیگری فرستاده شود حدود ۶ فریم دیرتر به مقصد می‌رسد، اگر این پیغام جوابی هم لازم داشته باشد این دوبرابر می‌شود، یعنی باز ۶ فریم دیگر طول می‌کشد تا جواب به مقصدش برسد.



در بازی‌های اکشن زمان عکس‌العمل خیلی مهم است. بازی‌کنان حرفه‌ای معمولاً می‌توانند به رویدادهای مختلف بازی در عرض چند صدم ثانیه عکس‌العمل نشان دهند. آن‌ها به این تأخیرها بسیار حسّاس می‌شوند. گلوله‌ای را در نظر بگیرید که با سرعت ۵۰۰ پیکسل بر ثانیه حرکت می‌کند. این گلوله در مدّت ۲۰۰ میلی‌ثانیه به اندازه‌ی ۱۰۰ پیکسل حرکت می‌کند که مسافت آن‌چنان کمی نیست.

Latency باعث به وجود آمدن مشکلات وحشتناکی می‌شود. برای این‌که بازی منصافه باشد، باید همه‌ی بازی‌کنان یک چیز یکسان را ببینند و برای عکس‌العمل نشان دادن به رویدادهای بازی فرصت برابری داشته باشند. در عمل هر بازی‌کنی با یک تأخیر زمانی بازی را انجام می‌دهد و این هاست است که برای اتّفاقات اصلی بازی تصمیم می‌گیرد. با این حال تکنیک‌هایی هستند که برای جبران این تأخیر زمانی استفاده می‌شوند و بازی را منصفانه نگه می‌دارند.

Packet Delay Variance (PDV)

مشکل دیگر Latency این است که می‌تواند برای هر پیامی متفاوت باشد. اگر دو پیام از طریق اینترنت ارسال شوند، یکی از آن‌ها می‌تواند در عرض ۵۰ میلی‌ثانیه فرستاده شود، و دومی در عرض ۷۰ میلی‌ثانیه. به این مشکل packet delay variance یا به اختصار پی‌دی‌وی (PDV) گفته می‌شود. در کانستراکت ۲ پی‌دی‌وی از تفاضل کمترین Latency از بیشترین Latency به دست می‌آید، در مثالی که الآن زدیم مقدار پی‌دی‌وی ۲۰ میلی‌ثانیه بود. پی‌دی‌وی زیاد از Latency زیاد بدتر است. موتور چندنفره‌ی کانستراکت ۲ پیش‌بینی می‌کند که چه قدر طول می‌کشد تا پیام‌ها برسند و سعی می‌کند تا تأخیر پیام‌ها را جبران کند. Latency زیاد و PDV کم یعنی پیش‌بینی‌ها به خوبی عمل می‌کنند و تأخیرها به درستی جبران می‌شوند، در نتیجه گیم‌پلی پایداری داریم. Latency کم و PDV زیاد یعنی پیام‌ها به طور غیر قابل پیش‌بینی دریافت می‌شوند، و اصلاحاتی که موتور چندنفره برای جبران تأخیر انجام می‌دهد، نادرست از آب در می‌آیند، در نتیجه شاهد گیم‌پلی بی‌ثباتی خواهیم بود.

Packet Loss

ممکن است پیام‌ها به طور کامل از دست بروند. در حالی که دستگاه‌های روتر نهایت تلاش خود را می‌کنند تا پیام‌ها را عبور دهند، اگر سیلابی از تعداد بسیار زیادی پیام به سمتشان بیاید، حافظه‌ی‌شان کم بیاورد، کرش کنند، برقشان برود و غیره اجازه دارند پیام‌ها را دور بریزند. Packet loss درصد پیام‌هایی است که از دست می‌روند. موتور چندنفره به راحتی می‌تواند بسته‌هایی را که به طور اتّفاقی از دست رفته‌اند جبران کند. امّا اگر packet loss خیلی زیاد باشد، گیم‌پلی دچار مشکل خواهد شد. اگر حجم اطّلاعات روتر بخواهد از فضایی که دارد فراتر برود، بعضی از پیام‌ها پردازش خواهند شد و دوباره فرستاده می‌شوند، و بقیّه‌ی پیام‌ها دور انداخته می‌شوند. پیام‌های کوتاه‌تر فرستاده می‌شوند و پیام‌های بزرگ‌تر از بین می‌روند. اگر روترها این کار را نکنند مشکلاتی برایشان پیش می‌آید که لازم نمی‌بینم در این‌جا درباره‌اش حرفی بزنم.

پهنای باند

پهنای باند به سرعت ارسال اطّلاعات ربط دارد. همه‌ی اتّصالات اینترنتی دارای یک حدّاکثر سرعت دانلود و آپلود هستند، که معمولاً در واحد مگابیت بر ثانیه اندازه‌گیری می‌شود. چون هر یک بایت هشت بیت است، اگر سرعتی بر حسب مگابیت بر ثانیه را به شما بدهند با تقسیم آن بر هشت سرعت را بر حسب مگابایت بر ثانیه به دست خواهید آورد (چون اکثر سیستم‌های نرم افزاری، همچنین کانستراکت ۲، اطّلاعات را بر حسب بایت اندازه می‌گیرند نه بیت). مثلاً، یک اتّصال با پهنای باند ۴۰ مگابیت بیت بر ثانیه از نظر علمی در هر ثانیه ۵ مگابایت اطّلاعات را انتقال می‌دهد. با این وجود سرعت آپلود می‌تواند فرق داشته باشد، و معمولاً بیشتر مصرف اینترنت صرف دانلود می‌شود و آپلود مصرف کم‌تری دارد.

پهنای باند می‌تواند به عنوان دهانه‌ی بطری برای بازی‌های بزرگ محسوب شود. وقتی تعداد بیشتری پیر به بازی ملحق می‌شود، برای این‌که همه‌ی پیرها به روز باشند هاست مجبور می‌شود حجم بیشتری را آپلود کند. بالأخره هاست مجبور می‌شود با سرعتی اطّلاعات را آپلود کند که سرعت آپلودش اجازه نمی‌دهد، در نتیجه Packet loss رخ خواهد داد. این دلیل دیگری است برای فرستادن پیام‌های کوتاه با پایین‌ترین سرعت قابل قبول، چون این کار استفاده از پهنای باند را به حدّاقل می‌رساند.

مبارزه با شرایط نامطلوب شبکه

برای به حدّاقل رساندن مشکل پهنای باند و packet loss، موتور چندنفره داده‌ها را حدّاکثر تا ۳۰ بار در ثانیه انتقال می‌دهد. داده‌ها حدّاکثر ۳۰ بار در ثانیه، یعنی نصف نرخ فریم بازی در حالت عادی، منتقل می‌شوند.

برای پیرها

به خاطر این‌که اطّلاعاتی مثل مختصّات اشیاء، فریم در میان ارسال می‌شوند، ممکن است بازی تکّه‌تکّه اجرا شود و اشیاء نامنظم حرکت کنند، و مشکلاتی با PDV و packet loss داشته باشیم.

برای بهبود گیم‌پلی، موتور چندنفره بین مقادیر ارسالی را درون‌یابی می‌کند (مقدار بین دو داده‌ی متوالی مثل مختصّات X و Y را حدس می‌زند). این درون‌یابی در مورد مکان اشیاء به صورت خطّی انجام می‌شود. برای مثال در یکی از فریم‌های بازی مختصّات شیء دریافت می‌شود، در فریم بعدی مختصّاتی دریافت نمی‌شود و در فریم بعد از آن مختصّات جدید دریافت می‌شود، در آن فریمی که مختصّاتی دریافت نشد، موتور چندنفره میانگین آن دو مختصّات دیگر را به شیء نسبت می‌دهد. در نتیجه بدون نیاز به هیچ داده‌ی بیشتری باز هم حرکت شیء مورد نظر نرم خواهد بود.

امّا موقع رندر شدنِ[۴] یک فریم درون‌یابی شده، موتور چندنفره هنوز نمی‌داندکه مقدار بعدی چه چیزی است، چون آن مقدار در آینده خواهد رسید! به جای استفاده از ماشین زمان، موتور چند نفره این مشکل را با **افزودن یک تأخیر ۸۰ میلی‌ثانیه‌ای برای پیرها** مرتفع کرده‌است. این یعنی موتور چندنفره از چند به روز رسانی قبل از این‌که رندر شوند پیشاپیش باخبر است. حتّی اگر به خاطر packet loss به روز رسانی بعدی دریافت نشود، موتور چندنفره از به روزرسانی‌های بعد از آن استفاده کرده و چند فریم را پشت سر هم درون‌یابی می‌کند. حتّی اگر وضعیّت شبکه آن قدر خراب باشد که برای ۸۰ میلی‌ثانیه هیچ به‌روزرسانی‌ای دریافت نشود، موتور چندنفره از ۲ به‌روزرسانی قبلی کمک گرفته و شیء را در همان راستا در فریم جدید حرکت می‌دهد. البتّه این یک کار حدسی است، ممکن است در همان زمان آبجکت ناپدید شده و در مکانی دیگر ظاهر شود، و درون‌یابی اشتباه شود.

برای درون‌یابی مقادیر، موتور چندنفره به شما اجازه می‌دهد تا یکی از سه حالت زیر را انتخاب کنید: خطّی[۵] (مثل مختصّات اشیاء)، زاویه‌ای[۶] (مثل زاویه‌ی اشیاء)، و هیچ[۷] (برای داده‌هایی که نباید درون‌یابی شوند، مثل یک مقدار بولی که نشان می‌دهد آیا لیزر روشن است یا نه)

بی‌ثباتیِ Latency

همان‌طور که گفتیم Latency می‌تواند تغییر کند. گاهی اوقات Latency در یک دقیقه‌ی اوّل تدریجاً بهبود می‌یابد یا پس از اتّصال، به تدریج اتّصال بهینه‌سازی می‌شود و روترها سریع‌ترین مسیر ممکن را تشخیص می‌دهند. Latency به طور ثابت اندازه‌گیری می‌شود تا چنین بی‌ثباتی‌هایی تشخیص داده شوند.

اگر Latency کاهش پیدا کند، موتور چندنفره به طور موقّت بازی را کمی سریع‌تر به جلو پیش می‌برد، زیرا کاهش Latency باعث می‌شود که سریع‌تر وقایع بازی را ببینید. از طرف دیگر، اگر Latency افزایش پیدا کند، سرعت حرکت اشیاء بازی کمی کاهش می‌یابد، زیرا افزایش Latency باعث می‌شود وقایع بازی را دیرتر ببینید. این کار برای جلوگیری از تکّه‌تکّه اجرا شدن بازی و این‌که تأخیر بیش از ۸۰ میلی‌ثانیه باعث حرکت نامنظّم اشیاء نشود، ضروری است. این کارها باید به قدری نرم انجام شوند که تقریباً نامحسوس باشند. این ویژگی‌ها باعث می‌شوند که مطمئن شویم در شرایط مختلف اتّصال بهترین عملکرد را دارا خواهیم بود.

برای هاست

هاست نسخه‌ی معتبر بازی را دارد. چون هاست نمی‌خواهد به خودش اطّلاعاتی ارسال کند، Latency اش صفر است. به عبارت دیگر، هاست در بازی «واقعی» شرکت می‌کند، و پیرها تمام تلاششان را می‌کنند تا بازی را همان طور ببینند که هاست دارد می‌بیند. در نتیجه هاست از مزیّت یک گیم‌پلی خوب برخوردار است.

شاید فکر کنید که هاست هم نیاز دارد تا بین به‌روزرسانی‌هایی که از پیرها دریافت می‌کند، درون‌یابی کند، ولی در حقیقت چنین چیزی اتّفاق نمی‌افتد، چون پیرها نباید به هاست بگویند که کجا هستند. در حقیقت این هاست است که به پیرها می‌گوید باید کجا باشند. دلیل این کار در قسمت بعدی بیان شده است.

---

[۴] رندر شدن = نمایش تصویر روی صفحه‌ی نمایش

[۵] linear

[۶] angular

[۷] none

جلوگیری از تقلّب

اگر پیرها به هاست بگویند که کجا هستند و دارند چه کاری انجام می‌دهند، و هاست به آن‌ها اعتماد کند، می‌توانند تقلّب کنند. می‌توانند ناگهان اطّلاعاتی ارسال کنند که در طرف دیگر دیوار قرار دارند، یا بگویند بازی‌کنی را که در حقیقت به آن نرسیده‌اند را زده‌اند، و... برای کاهش تقلّب، پیرها به هاست فقط ورودی‌هایشان را می‌گویند، ورودی‌هایی مثل این‌که در حال حاضر فلان کلید جهتی را نگه داشته‌اند. هاست این ورودی‌ها را دریافت می‌کند و شروع به حرکت دادن آن بازی‌کن می‌کند، و مکانی که بازی‌کن در آن قرار دارد را به او ارسال می‌کند.

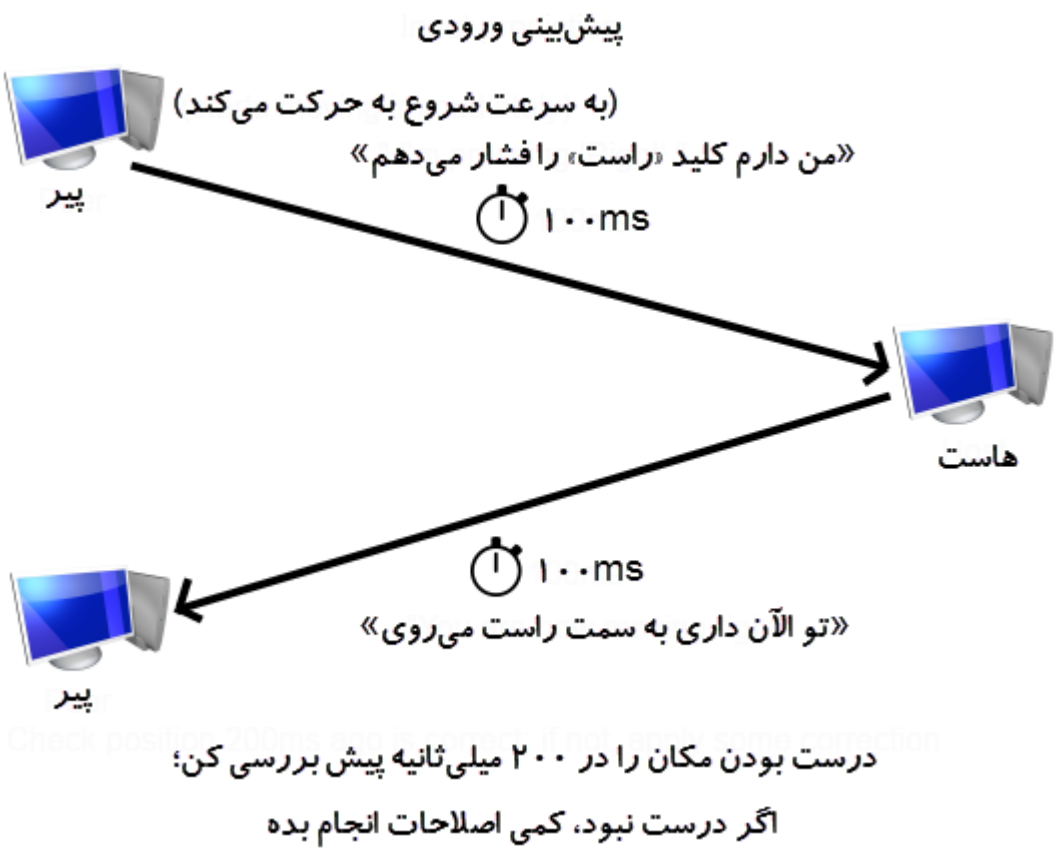
با این کار یک مشکل به وجود می‌آید و آن تأخیر در ورودی‌های بازی‌کن‌هاست. اگر Latency بازی ما ۱۰۰ms باشد، و یک بازی‌کن کلید «چپ» را فشار دهد ۱۰۰ms طول می‌کشد تا این خبر به هاست برسد. بعد ۱۰۰ms دیگر طول می‌کشد تا هاست بازی‌کن را از مکان جدیدش مطّلع کند، این یعنی همه‌ی کنترل‌های پیر خیلی با تأخیر انجام می‌شوند، کسری از ثانیه طول می‌کشد تا ورودی‌هایی که پیر وارد کرده است عمل کنند. برای حلّ این مشکل بازی‌ای که پیر می‌بیند (بازی محلّی) از *پیش‌بینی ورودی محلّی*[۸] استفاده می‌کند.

پیش‌بینی ورودی محلّی

با فشرده‌شدن کلید چپ توسّط بازی‌کن، پیغامی در این مورد به هاست فرستاده می‌شود، بعد در بازی محلّی، بازی‌کن به سمت چپ حرکت می‌کند، بدون این‌که منتظر تأیید هاست بماند! از آن‌جایی‌که هاست و پیر، هر دو دارند بازی یکسانی را انجام می‌دهند، معمولاً بازی‌شان خیلی زیاد با هم تطابق دارد. حالا کنترل‌های بازی‌کن با تأخیر مواجه نمی‌شوند، والبتّه پیرها هنوز نمی‌توانند تقلّب کنند؛ آن‌ها فقط ورودی‌هایشان را ارسال می‌کنند، و نمی‌توانند وانمود کنند که در جایی دیگر قرار دارند. حتّی اگر بازی محلّی خودشان را هک کنند بی فایده خواهد بود، چون با این کار آن‌ها خودشان را در مکان نادرستی در مقایسه با هاست می‌بینند.

در این صورت بازی‌کن پیر قطعاً خود را در محلّی نمی‌بیند که هاست دارد او را در آن محل می‌بیند. این باعث به وجود آمدن خطاهای کوچک بسیاری از جمله در تنظیم زمان شبکه، هماهنگ‌سازی نرخ فریم، اندازه‌گیری دلتا تایم (Δt) و... می‌شود. البته پیر هنوز پیام‌هایی را از هاست دریافت می‌کند که به او می‌گویند در کجا قرار دارد، امّا این پیام‌ها با تأخیر می‌رسند، در مثال قبلی، دیدیم که این پیام ۲۰۰ میلی‌ثانیه بعد به پیر می‌رسد. برای تصحیح این مشکلات، موتور چندنفره با استفاده تاریخچه‌ی شیء نگاه می‌کند که شیء مزبور در ۲۰۰ میلی‌ثانیه پیش در مقایسه با مکان دریافتی از هاست در چه مکانی قرار داشته است. به عبارت دیگر، اصلاحات با توجّه به این‌که شیء در گذشته (با محاسبه‌ی Latency) در چه مکانی قرار داشته است انجام می‌شوند، اگر شیء در مکان نادرستی بود، موتور چندنفره آن را به تدریج حرکت می‌دهد تا در مکان صحیح قرار بگیرد. این کار به آرامی انجام می‌شود تا کمتر به چشم بیاید. ولی در مواردی که اختلاف فاصله خیلی زیاد است، یک اصلاح بزرگ که به چشم بیاید ضروری است.

[۸] Local input prediction



این ویژگی فقط به مکان اشیاء محدود نمی‌شود. موتور چندنفره این توانایی را دارد که تمام این هماهنگ‌سازی‌ها را برای هر متغیّر اینستنسی هم انجام دهد، نه فقط مختصّات X و Y.

Lag

وضعیّت خراب شبکه می‌تواند تأثیرات آشکاری بر گیم‌پلی داشته باشد. گیمرها معمولاً چنین مشکلاتی را به «lag» می‌شناسند. این مشکلات شامل موارد زیر هستند: حرکت ناهموار (وقتی تا ۸۰ میلی‌ثانیه داده‌ها نرسند)، پرش یا ناپیوستگی در گیم‌پلی (وقتی packet loss رخ دهد و باعث شودبه‌روزرسانی مهمّی که وضعیّت بازی را به طور قابل توجّهی تغییر می‌دهد از دست برود)، تغییر مکان ناگهانی بازی‌کن (وقتی پیش‌بینی ورودی محلّی با یک خطای بزرگ مواجه شود)، یا اتّفاقات به ظاهر غیر عادلانه‌ای که پیش می‌آید (این مشکل معمولاً نتیجه‌ی این است که بازی‌کن‌ها بازی را با تأخیرهای مختلفی مشاهده می‌کنند، و هاست به نفع یک پیر خاص تصمیم‌گیری می‌کند).

متأسّفانه راه حلّی برای این‌که همیشه از شرّ این مشکلات خلاص شویم وجود ندارد، مگر این‌که اتّصال اینترنت بهتری پیدا کنیم. این مشکلات معمولاً تحت شرایط سختی از قبیل packet loss کامل به مدّت چند ثانیه، یا تغییرات ناگهانی و شدید در latency یا PDV به وجود می‌آیند. در بعضی مواقع، برای جبران کامل lag استفاده از تکنیک‌های مختلف غیر ممکن است. گاهی اوقات این واقعیّت ارتباط اینترنتی است.

جبران Lag

مشکل زیر را در نظر بگیرید: بازی‌کن الف ایستاده است و بازی‌کن ب را هدف قرار داده است. بازی‌کن الف موس را دقیقاً روی بازی‌کن ب می‌برد و شلّیک می‌کند. (فرض کنید که تیر مثل عکس زیر درجا برخورد کند، مثلاً لیزر باشد و یا گلوله‌ای با سرعت

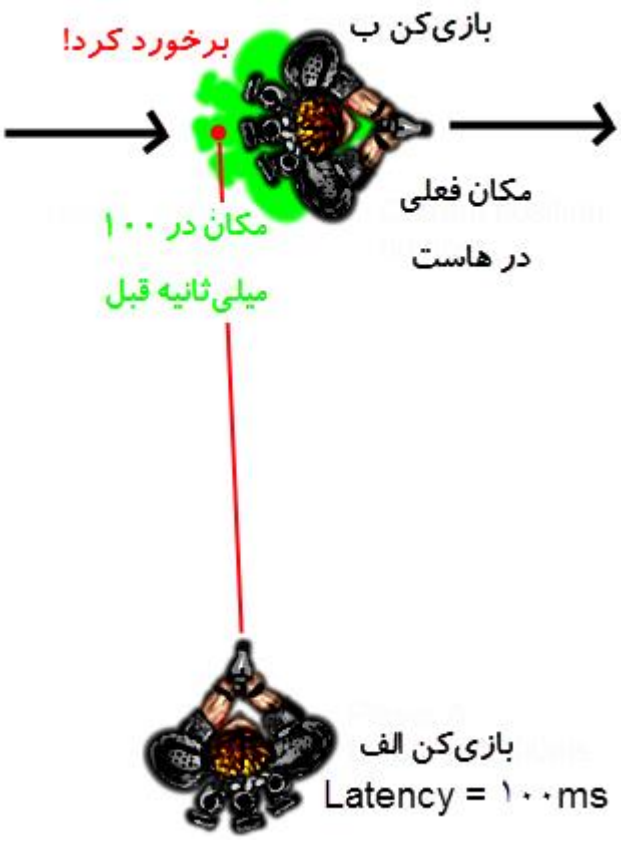
بی‌نهایت.) در این حین که دارد خبر این شلّیک به هاست می‌رسد (فرض کنید ۱۰۰ میلی‌ثانیه طول بکشد)، بازی‌کن ب به سمت راست حرکت می‌کند. وقتی خبر شلّیک بعد از ۱۰۰ میلی‌ثانیه به هاست می‌رسد، هاست بررسی می‌کند که آیا برخوردی صورت گرفته‌است یا نه، امّا برخوردی را تشخیص نمی‌دهد، چون بازی‌کن ب دیگر در آن مکان قرار ندارد! بازی‌کن الف می‌بیند که گلوله‌اش به بازی‌کن ب برخورد کرده است، ولی بازی‌کن ب هیچ صدمه‌ای نمی‌بیند.

در کل به خاطر Latency هر کاری را که بازی‌کن‌ها انجام می‌دهند، هاست یک ذرّه دیرتر می‌بیند، و در نهایت بازی‌کن‌ها اهداف خود را از دست می‌دهند. این نوع مشکلات بازی‌کن‌هایی را که در نهایت یاد می‌گیرند باید فضای خالی جلوی بازی‌کنی که در حال حرکت است را هدف بگیرند، نه خود بازی‌کن را، حسابی عصبانی می‌کند. این تا حدّی واقعی بودن بازی را از بین می‌برد و آن را ناعادلانه می‌کند. جبران Lag تکنیکی است برای جلوگیری از وقوع این نتیجه.

به عقب بردن زمان

هاست می‌داند که چه قدر طول می‌کشد تا هر بازی‌کن بازی را ببیند، زیرا مقدار Latency برای تک تک آن‌ها محاسبه می‌شود. بنابراین وقتی به هاست پیغام می‌رسد که بازی‌کن الف شلّیک کرده است، هاست می‌داند که این شلّیک ۱۰۰ میلی‌ثانیه قبل اتّفاق افتاده است.

هاست تاریخچه‌ی هر شیء را در چندثانیه‌ی اخیر به یاد دارد. پس می‌تواند نگاه کند که بازی‌کن ب در ۱۰۰ میلی‌ثانیه‌ی پیش در چه مکانی قرار داشته است. بعد بررسی می‌کند که آیا دقیقاً در همان لحظه‌ی اصلی که شلّیک انجام شد به بازی‌کن ب برخورد کرده است یا نه، و این یعنی مشکل ما برطرف شد.



ولی این کار باعث به وجود آمدن یک مشکل دیگر می‌شود. بازی‌کن ب نیز بازی را با تأخیر دریافت می‌کند. یعنی می‌بیند که بازی‌کن الف دقیقاً او را هدف نگرفته‌است و شلّیک می‌کند و به او برخورد می‌کند. در بعضی اوقات وضع بدتر هم می‌شود، مثلاً بازی‌کن ب جایی را پیدا می‌کند و پشت آن پناه می‌گیرد تا بازی‌کن الف نتواند مستقیماً به او شلّیک کند، ولی با شلّیک بازی‌کن الف او آسیب می‌بیند. این موضوع برای بازی‌کن ب آزاردهنده است، ولی هرچه باشد به بدی موقعی نیست که هیچ اصلاحی انجام نشود. لااقل بازی‌کن‌ها برای این‌که بتوانند به هم آسیب برسانند مجبورند مستقیماً یکدیگر را هدف قرار دهند. راه حلّ رایج این مشکل این است که به صورت خیلی دقیق نشان ندهیم که بازی‌کن‌ها کجا را هدف گرفته‌اند، بنابراین بازی‌کن ب به راحتی نمی‌تواند بفهمد که بازی‌کن الف چگونه او را هدف قرار داده است، و برخورد گلوله به او قابل قبول‌تر خواهد بود.

چند بازی‌کن می‌توانند به یک بازی ملحق شوند؟

تعداد بازی‌کن‌هایی که می‌توانند به بازی ملحق شوند به پهنای باند آپلود هاست بستگی دارد. موتور چندنفره محدودیّتی را تحمیل نمی‌کند، ولی در عمل با محدودیّت مواجه خواهیم شد.

مشکل اینجاست که هاست اطّلاعاتی را که از n بازی‌کن دریافت کرده است، می‌خواهد به هرکدام از دیگر بازی‌کن‌ها بفرستد. برای مثال اگر هاست بخواهد ۱۶ بایت داده به هر بازی‌کن ارسال کند، ابتدا باید از هرکدام از بازی‌کن‌ها ۱۶ بایت داده دریافت شود که اگر تعداد بازی‌کن‌ها n تا باشد n×۱۶ بایت داده دریافت می‌شود. حالا این داده‌ها قرار است به همه‌ی بازی‌کن‌ها (n بار) ارسال شود، در نتیجه عدد قبلی n برابر شده و هاست باید ۱۶×n×n بایت داده ارسال کند. مثلاً:

۱۰ بازی‌کن = ۱۶×۱۰×۱۰ = ۱۶۰۰ بایت برای هر بروزرسانی

۲۰ بازی‌کن = ۱۶×۲۰×۲۰ = ۶۴۰۰ بایت برای هر بروزرسانی

۳۰ بازی‌کن = ۱۶×۳۰×۳۰ = ۱۴۴۰۰ بایت برای هر بروزرسانی

...

۱۰۰ بازی‌کن = ۱۶×۱۰۰×۱۰۰ = ۱۶۰۰۰۰ بایت برای هر بروزرسانی

اگرچه تعداد بازی‌کن‌ها معمولی بالا می‌رود، امّا نیاز به پهنای باند به توان۲ برابر افزایش می‌یابد. یعنی حتّی با داشتن یک سرور خیلی قدرتمندتر یا داده‌های کم‌حجم‌تری برای هر بازی‌کن، نمی‌توانید تعداد بازی‌کن‌ها را بیش از حد زیاد کنید.

به طور پیش‌فرض به‌روزرسانی‌ها ۳۰ بار در یک ثانیه ارسال می‌شوند، پس در مثال قبل با ۱۰۰ بازی‌کن، عملاً هاست باید دارای پهنای باندی در حدود ۵ مگابایت بر ثانیه (یا ۴۰ مگابیت بر ثانیه) باشد. این مقدار برای یک اینترنت خانگی بسیار زیاد است، امّا برای یک سرور اختصاصی خیر.

گذشته از این، هاست باید بتواند بازی را برای تعداد زیادی بازی‌کن اجرا کند و وظایفی مثل جبران lag و بررسی‌های برخورد را انجام دهد که باعث می‌شود شدیداً از CPU استفاده شود. درضمن اگر هاست یکی از بازی‌کن‌ها باشد، بازی باید برایش رندر هم بشود. معمولاً مقدار کلّ کاری که باید CPU انجام دهد با افزایش تعداد بازی‌کن‌ها به سرعت زیاد می‌شود، اگرچه حدّاکثر تعداد بازی‌کن‌ها بسته به نوع بازی و اتّصال اینترنت متفاوت است، ولی تقریباً پیوستن بیش از ۱۰۰ بازی‌کن به بازی غیر ممکن است.

تنظیم فرمت به‌روزرسانی‌ها

این امکان وجود دارد که نوع دقیق مقادیری را که می‌خواهید موتور چندنفره آن‌ها را منتقل کند تعیین کنید. استفاده‌ی صحیح از این ویژگی باعث می‌شود نیاز کمتری به پهنای باند داشته باشیم، بدون این‌که تغییر خاصّی در گیم‌پلی رخ دهد، و حتّی حدّاکثر تعداد بازی‌کن‌هایی که به بازی ملحق می‌شوند را افزایش دهیم.

اگر با بیت (bit)، بایت (byte) و باینری (binary) آشنایی ندارید، می‌توانید مقاله‌های بایت و باینری را در ویکی‌پدیا مطالعه کنید و یا در این باره در گوگل جستجو کنید. به طور خلاصه، در کامپیوتر تمام اعداد در مبنای ۲ (دستگاه باینری) نگه‌داشته می‌شوند: در این مبنا هر رقم فقط می‌تواند ۰ و یا ۱ باشد. یک بایت ۸ بیت است، پس می‌تواند $۲^۸$ (۲۵۶) مقدار مختلف داشته باشد، یا مقداری از ۰ تا ۲۵۵. بایت‌های بیشتر می‌توانند محدوده‌ی وسیع‌تری از اعداد را در خود نگه دارند. بعضی از انواع اعداد اعشاری مثل ۰٫۳ حدّاقل ۴ بایت را اشغال می‌کنند.

نوع مقادیری که در کانستراکت ۲ می‌توانید استفاده کنید به شرح زیر است:

High (double، ۸ بایت): یک عدد double-precision floating-point که می‌تواند اعداد اعشاری را با دقّت زیاد در حدود ۱۵ تا ۱۷ رقم اعشار در خود نگه‌دارد. ولی به تنهایی برای یک عدد ۸ بایت حجم می‌گیرد، که می‌تواند استفاده از پهنای باند را بی‌خودی افزایش دهد. در عمل، انتخاب این نوع مقدار بعید است، مگر در مواقعی که واقعاً ضروری باشد.

Normal (float، ۴ بایت): یک عدد single-precision floating-point که می‌تواند اعداد اعشاری را با دقّت حدود ۶ تا ۹ رقم اعشار را در خود نگه دارد. یعنی بعضی از اعداد به اعدادی با تعداد اعشار کم‌تر گرد می‌شوند. با این‌حال در عمل، استفاده از این نوع مقدار به صرفه‌تر از double است، چون حجمی نصف double دارد و ارقام اعشاری بعد از رقم ششم معمولاً اهمّیّتی ندارند (مثلاً ۰٫۳۳۳۳۳۳ به اندازه‌ی کافی دقیق هست و واقعاً نیازی به ۰٫۳۳۳۳۳۳۳۳۳۳۳۳۳۳۳ نداریم.)

Low (int16، ۲ بایت): یک عدد صحیح ۱۶ بیتی که فقط می‌تواند در محدوده‌ی ۳۲۷۶۸- تا ۳۲۷۶۷ قرار داشته باشد.

Very low (uint8، ۱ بایت): یک عدد حسابی ۸ بیتی که فقط می‌تواند در محدوده‌ی ۰ تا ۲۵۵ قرار داشته باشد.

برای کاهش میزان مصرف پهنای باند، باید تا حدّ امکان کم‌ترین دقّتی را که اثر مهمّی در گیم‌پلی ایجاد نکند انتخاب کنیم. مثلاً برای مختصّات X و Y دقّت low (int16) مناسب است. زیرا معمولاً اندازه‌ی لیوت بیشتر از ۳۲۶۷×۳۲۶۷ نیست، و اعشارِ مختصّات پیکسل‌ها اثری در گیم‌پلی ندارد. این دقّت کاملاً کافی است و نسبت به double چهاربرابر کم‌تر از پهنای باند استفاده می‌کند.

دقّت low (int16) و Very low (uint8) برای ورودی‌هایی که پیرها وارد می‌کنند و به هاست فرستاده می‌شود مفید است. با استفاده از اکسپرشن‌های setbit، getbit و togglebit در کانستراکت ۲ می‌توانید بیت‌هایی جداگانه را در این اعداد قرار دهید. اگر یک بازی از ۴ کلید جهتی برای حرکت و از Space برای شلّیک استفاده کند، فقط ۵ ورودی وجود دارد، که همه‌اش می‌تواند در یک ۸ بیت ذخیره شود با این حساب که صفر نشان دهنده‌ی این است که کلید مورد نظر در حال فشار داده شدن نیست و یک نشان‌دهنده‌ی این است که کلید مورد نظر دارد فشار داده می‌شود. بعد هاست می‌تواند به هرکدام از بیت‌ها نگاه کند و ورودی مناسب با آن را شبیه‌سازی کند. همه‌ی این‌ها فقط یک بایت حجم می‌گیرند. اگر شما بیشتر از ۸ ورودی دارید می‌توانید از یک مقدار

int16 استفاده کنید، با این‌کار در استفاده از پهنای باند خیلی صرفه‌جویی می‌کنید. به طریق مشابه، هاست هم می‌تواند چندین حالت روشن/خاموش[۹] را در قالب یک مقدار به پیرها بفرستد.

پلاگین Multiplayer در کانستراکت ۲

طرّاحی کلّی یک بازی چندنفره در کانستراکت ۲ چیزی شبیه این است.

روند یک بازی چندنفره

چرخه‌ی یک بازی چندنفره این‌چنین است.

۱. اتّصال به سیگنالینگ سرور و ورود به آن

۲. پیوستن به یک اتاق

۳. در طول بازی پیرهای دیگر می‌توانند به بازی بپیوندند یا از آن خارج شوند.

۴. اطّلاعات اشیاء مختلف و پیام‌های دیگری مثل چت بین پیرهای متّصل‌شده رد و بدل می‌شود.

۵. سرانجام بعد از اتمام بازی، همه اتاق را ترک می‌کنند، و شاید وارد یک اتاق دیگر شوند (برگشت به گام ۲)

توجّه کنید که اگر اتّصال هاست قطع شود، بازی تمام می‌شود. چون هاست به منزله‌ی سرور بازی عمل می‌کند با قطع شدن اتّصال آن بازی نمی‌تواند ادامه پیدا کند. به همین دلیل بازی‌های چندنفره‌ی آنلاین، مخصوصاً بازی‌های بزرگ با تعداد بازی‌کن زیاد، معمولاً در هاست اختصاصی روی یک سرور مرکزی اجرا می‌شوند. امّا این مشکل برای بازی‌های دونفره وجود ندارد، زیرا هرکدام از دو بازی‌کن که اتّصال‌شان قطع شود بازی خاتمه می‌یابد.

پلاگین Multiplayer دارای ویژگی‌هایی است که مرحله‌ی سیگنال‌دهی را انجام می‌دهد (گام‌های ۱ و ۲)، و نیز ویژگی‌های «room» گیم‌پلی را پوشش می‌دهند، مثلاً وقتی پیرها بازی را ترک کنند، به بازی ملحق شوند، پیامی دریافت شود و... به ما خبر می‌دهند.

طرّاحی کلّی

این طور در نظر گرفته شده است بازی‌ها طوری طرّاحی شده اند که یک آبجکت‌تایپ داریم که نماینده‌ی اشیاء تحت کنترل بازی‌کن‌هاست. حالا این شیء توسّط تمام بازی‌کن‌ها به طور یکسان استفاده می‌شود که شامل شما نیز می‌شود (حتّی وقتی هاست هستید). عاقلانه است اسم این شیء را Peer بگذارید، زیرا این شیء نماینده‌ی پیرها می‌باشد. اگر بازی‌کن‌های شما از ترکیب چند شیء ساخته شده‌اند، مثلاً از ترکیب یک اسلحه و یک بدن جدا، یکی را به عنوان شیء Peer اصلی در نظر بگیرید و آن را با سایر اشیاء مربوطه در یک کانتینر (Container) قرار دهید، این کار باعث می‌شود مطمئن شوید بازی‌کن شما به صورت کامل به وجود می‌آید و یا نابود می‌شود، و اشیاء مربوط به آن به صورت خودکار به شیء اصلی می‌چسبند، بنابراین تا حدّ زیادی می‌توانید شیء اصلی را به عنوان نماینده‌ی کلّ آن بازی‌کن تلقّی کنید.

هاست و پیرها همه پروژه‌ی یکسانی را اجرا می‌کنند. یعنی همان یک پروژه‌ی شما باید بتواند هم با ایونت‌های مربوط به هاستی سروکار داشته‌باشد که نسخه‌ی معتبر و اصلی را بازی می‌کند، و هم با ایونت‌های مربوط به پیرهایی که نسخه‌ی تأخیردار را بازی

[۹] منظور چیزهایی در بازی هست که نمی‌توانند بیش از دو حالت داشته باشند

می‌کنند و فقط ورودی‌هایشان را ارسال می‌کنند. راحت‌ترین روش برای این کار این است که دو ایونت گروپ یکی مختصّ هاست و یکی مختصّ پیرها داشته باشیم، و بعد از ملحق شدن به بازی و تشخیص این‌که آیا هاست شدیم یا پیر، فقط ایونت گروپ مناسب را فعّال کنیم.

اکشن Sync object شیء Multiplayer راه اصلی نشان‌دادنِ این است که می‌خواهید داده‌هایی در مورد شیئی خاص را از طریق شبکه ارسال کنید. این اکشن به هاست می‌گوید که اطّلاعات مربوط به آن اشیاء را به همه‌ی پیرها ارسال کند، و به پیرها می‌گوید تا منتظر اطّلاعات مربوط به آن اشیاء بمانند. وقتی پیرها به بازی ملحق می‌شوند و یا آن را ترک می‌کنند، موتور چندنفره به صورت خودکار اشیائی را که نماینده‌ی آن پیرها هستند به وجود می‌آورد و یا حذف می‌کند. (اکشن Sync object می‌تواند برای اشیائی که نماینده‌ی هیچ پیری نیستند هم به کار رود، ولی بیشتر برای پیرها استفاده می‌شود)

هر پیر شناسه‌ای منحصر به فرد دارد که از سیگنالینگ سرور دریافت کرده است، این شناسه که Peer ID نامیده می‌شود یک رشته است که در انتهای آن تعدادی کاراکتر تصادفی برای منحصر به فرد شدن اضافه شده است. به طور کلّی این شناسه‌ها برای این به کار می‌روند که بتوانیم به طور مداوم به بازی‌کنی خاص اشاره داشته باشیم، حتّی اگر نام مستعار خود را در بازی تغییر دهد. Peer ID باید در یک متغیّر اینستنس ذخیره شود تا بدانید کدام شیء نماینده‌ی پیر مورد نظر است. وقتی موتور چندنفره شیئی را برای یک بازی‌کن به وجود آورد مقدار اکسپرشن Multiplayer.PeerID به Peer ID آن شیء تغییر می‌کند، بنابراین این مقدار می‌تواند در یک متغیّر اینستنس مربوط به همان شیء ذخیره شود. برای این‌که شما فقط شیء پیر مربوط به خودتان را کنترل کنید به راحتی می‌توانید با بررسی این‌که آیا مقدار آن متغیّر اینستنس با اکسپرشن Multiplayer.MyID[10] برابر است یا نه این کار را انجام دهید.

حالت‌های معتبر

۳ حالت برای انتقال اطّلاعات وجود دارد.

مطمئن‌ترین حالت Reliable ordered است. در این حالت یک پیام به صورت زنجیروار فرستاده می‌شود: اگر این پیام در بین راه به هر دلیلی از بین برود، مجدّداً فرستاده می‌شود تا وقتی‌که اطمینان حاصل شود پیام به مقصد رسیده است. در این حالت ترتیب پیام‌ها نیز رعایت می‌شود، بنابراین پیام‌ها دقیقاً به همان ترتیبی که ارسال شده‌اند به مقصد می‌رسند. این خیلی مفید است، ولی سریع‌ترین حالت نیست: اگر یک پیام در بین راه از بین برود، مانع ارسال همه‌ی پیام‌های بعدی می‌شود تا این‌که به مقصد برسد. یعنی یک پیام تنها ممکن است Latency را چندین برابر آن چیزی که باید باشد بکند زیرا حدّاقل به اندازه‌ی یک زمان رفت و برگشت باید صبر کند تا بداند پیام رسیده‌است یا نه. به طور کلّی این حالت فقط مناسب پیام‌های کم‌بسامد مثل پیام‌های چت است.

حالت بعدی Reliable unordered است. در این حالت یک پیام به صورت زنجیروار آن‌قدر فرستاده می‌شود تا مطمئن شود به مقصد رسیده است. ولی ترتیب پیام‌ها در این حالت مهم نیست، این باعث می‌شود وقتی یک پیام در بین راه از بین می‌رود و مجدّداً فرستاده می‌شود پیام‌های بعدی را پشت سرش نگه ندارد، در این موقع این پیام بعد از پیام‌هایی به مقصد می‌رسد که دیرتر از آن ارسال شده‌اند. با این حال Latency برای رسیدن همان یک پیام هنوز هم چند برابر می‌شود. این حالت برای رویدادهایی از گیم‌پلی که ربطی به هم نداشته باشند مناسب است، مثل دری که باز می‌شود یا انفجاری که رخ می‌دهد.

سریع‌ترین حالت Unreliable است. در این حالت پیام فرستاده می‌شود و پیگیری هم نمی‌شود. اگر پیام در بین راه از بین برود هیچگاه به مقصد نمی‌رسد. در این حالت نیز مثل حالت قبل ترتیب پیام‌ها مهم نیست. معمولاً این حالت برای پیام‌های عادی مناسب

---

۱۰ این اکسپرشن Peer ID شما را برمی‌گرداند

است که می‌خواهید هرچه سریع‌تر به مقصد برسند و یا اگر قرار است دیر برسند بهتر است اصلاً نرسند. اگر یکی از این پیام‌ها در بین راه از بین برود، پیام بعدی با داده‌ای جدیدتر به سرعت به مقصد می‌رسد، بنابراین پیام از دست رفته آنچنان مهم نیست. موتور چند نفره برای اشیائی که Sync object شده اند و همچنین ویژگی‌های درون‌یابی و جبران، از این حالت استفاده می‌کند. پس سعی نکنید دوباره روی این حالت تنظیمشان کنید.

آماده برای رفتن

با این‌که پلاگین Multiplayer کانستراکت ۲ تعداد زیادی از جزئیّات فنّی مثل پیش‌بینی ورودی و اتّصال را برای شما انجام می‌دهد، باز هم خیلی مهم است که تئوری را هم بلد باشید تا بتوانید با توجّه به این‌که چه داده‌ای با چه دقّتی و در چه حالت معتبری فرستاده می‌شود انتخاب‌های مناسبی انجام دهید. امیدوارم این آموزش به اندازه‌ی کافی یک شمای کلّی به شما داده باشد تا شروع کنید به ساخت اوّلین بازی چندنفره‌تان، با اطّلاع از این‌که پشت صحنه چه اتّفاقاتی دارد می‌افتد و شما باید چه کارهایی را انجام دهید تا بهترین استفاده را از ویژگی‌های چندنفره‌ی کانستراکت ۲ بکنید. با این‌که این آموزش روی تئوری تمرکز داشت، آموزش‌های بعدی در عمل، چگونگی رسیدگی به مسائلی از قبیل پیش‌بینی ورودی و جبران lag را با استفاده از ویژگی‌های خاص پلاگین Multiplayer پوشش می‌دهند.

مجتبی قاسم‌زاده تهرانی

si2.ir

mojtaba@si2.ir

آبان ۱۳۹۳