به مناسبت هفته‌ی وحدت و میلاد پیامبر گرامی اسلام (ص)

اشتباهات رایج در استفاده از ایونت‌های کانستراکت ۲

WWW.SI2.IR

مجتبی قاسم‌زاده تهرانی

فهرست

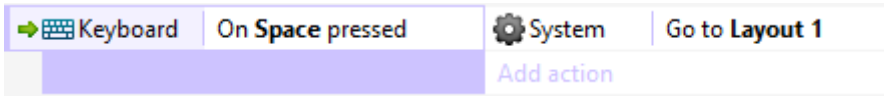
مقدّمه

سیستم ایونت کانستراکت ۲ به گونه‌ای طرّاحی شده که استفاده از آن حتّی برای تازه‌کاران ساده باشد. ما به خاطر ارتباطاتی که با کاربران داریم، متوجّه شدیم که بسیاری از آن‌ها چیزهایی را اشتباه فهمیده‌اند، یا به اشتباه، یا بی‌خودی از چیزی استفاده می‌کنند که لازم نیست. همچنین بعضی از اشتباهات به خاطر این است که درکی از نحوه‌ی عملکرد کامپیوتر ندارند، مثلاً این حقیقت را درک نمی‌کنند که حافظه یا قدرت پردازش کامپیوتر بی‌نهایت نیست! در این مقاله چند مثال به همراه توضیح بیشتر آورده‌ایم، پس سعی کنید این اشتباهات را در پروژه‌ی خود مرتکب نشوید!

کاندیشن Every tick اضافی

طبق اصول فنّی، کاندیشن Every tick فقط به معنی 'true' (درست) است. روش اجرای ایونت‌ها به این‌گونه است که فقط وقتی اکشن‌هایشان را اجرا می‌کنند که 'true' باشند. بنابراین، افزودن Every tick به هر ایونتی که دارای کاندیشن دیگری است هیچ تأثیری ندارد. در ایونت زیر کاندیشن Every tick زاید است:

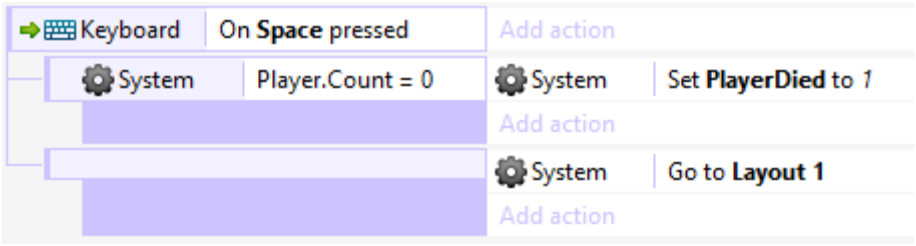
بهتر است به گونه‌ای از ایونت‌ها استفاده کنیم که در حدّ امکان خلاصه باشد، پس باید این کاندیشن حذف شود:



این ایونت دقیقاً مثل ایونت قبلی عمل می‌کند، و از آن ساده‌تر است.

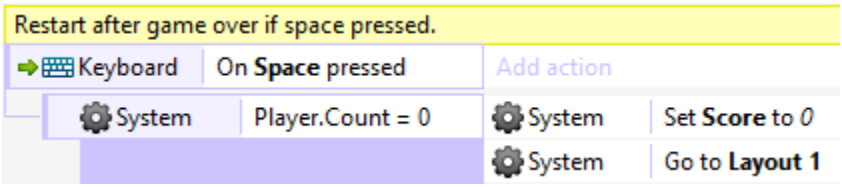
دلیل نام‌گذاری این کاندیشن به Every tick این است که دستورات ایونت شیت در هر تیک (زمان نمایش یک فریم، که معمولاً یک شصتم ثانیه است) یک بار بررسی می‌شوند. ایونتی که فقط شامل یک کاندیشن Every tick باشد، به خودی خود در هر تیک اجرا می‌شود، پس این نام‌گذاری دقیق است و به ما می‌گوید این کاندیشن برای چه کاری است. امّا اگر مثل مثال اوّل بی‌خودی از آن استفاده کنیم ممکن است کمی گمراه‌کننده شود، زیرا قطعاً اکشن‌ها در هر تیک یک‌بار اجرا نمی‌شوند!

ایونتی که هیچ کاندیشنی نداشته‌باشد نیز در هر تیک اجرا می‌شود، امّا این برای یک تازه‌کار کمی مبهم است، به همین دلیل توصیه می‌کنیم از کاندیشن Every tick استفاده کنید، مگر این‌که بخواهید در قالب ساب‌ایونت اکشن‌هایی را بعد از یک ساب‌ایونت دیگر اجرا کنید، در این مورد یک ایونت بدون کاندیشن مناسب‌تر است. مثال زیر این مورد را نشان می‌دهد: در این‌جا از ایونتی بدون کاندیشن استفاده کردیم، زیرا اکشن Go to Layout 1 باید بعد از بررسی کردن تعداد پلیرها اجرا شود. این کار باعث می‌شود فکر نکنیم این ایونت باید در هر تیک اجرا شود.

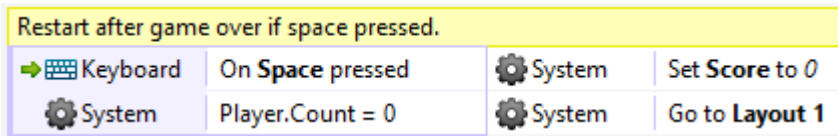


استفاده‌ی غیر ضروری از ساب‌ایونت

ساب‌ایونت‌ها برای تعریف لوجیک[۱] پیشرفته‌تری برای بازی عالی هستند. امّا معمولاً زیاد می‌بینیم که بی‌خودی از آن استفاده می‌شود. ساب ایونت‌ها، از بین اشیائی که ایونت والدشان[۲] پیک (Pick) کرده است، تعدادی را پیک می‌کنند، ساب‌ایونت معمولاً می‌تواند به راحتی با ایونت والد خود ترکیب شود. حالا مثالی را می‌بینیم:



در ایونت بالا، نیازی نیست که کاندیشن Player.Count = 0 را در یک ساب‌ایونت قرار دهید. اگر هر دو را در یک ایونت قرار دهید نتیجه هیچ فرقی نمی‌کند، مثل زیر:



ما توصیه می‌کنیم از این روش استفاده کنید تا ایونت راحت‌تر خوانده شود. در روش اوّل خواندن کمی سخت‌تر خواهد بود. در ضمن این طور استفاده از ساب ایونت این حس را القا می‌کند که وجودش به دلایلی ضروری است، که می‌تواند دیگران (یا خودتان) را گیج کند یا به اشتباه بیندازد. مخصوصاً اگر بخواهید به دیگران آموزش بازی‌سازی بدهید این مطلب خیلی مهم می‌شود ، چون ممکن است از شما بپرسند چرا از ساب‌ایونت استفاده کردید، درحالی‌که جواب خوبی برایش وجود ندارد.

از آن‌جایی‌که ساب‌ایونت‌ها به سازمان‌دهی ایونت‌ها کمک می‌کنند، عدّه‌ای از آن‌ها استفاده می‌کنند.[۳] این کار خوبیست، ولی توصیه می‌کنیم ترجیحاً از همین روش آسان خواندن استفاده کنید، و بعداً در صورت

---

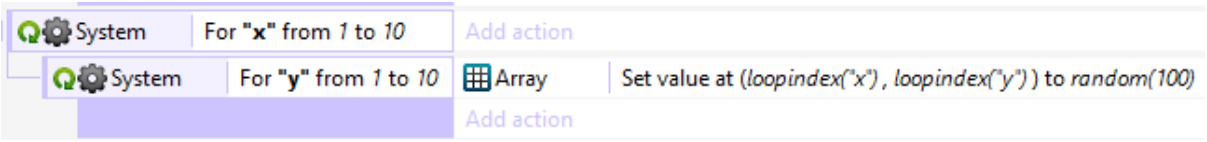
[۱] لوجیک (Logic): منطق، شرط‌ها و عمل‌هایی که در بازی برنامه‌ریزی می‌شوند.

[۲] ایونت والد (Parent event): به ایونتی گفته می‌شود که ساب ایونت مورد نظر زیرمجموعه‌ای از آن ایونت است.

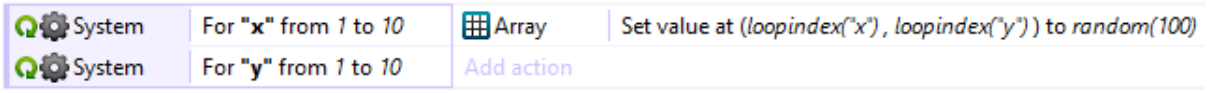
[۳] زیرا در کنار ایونت‌های والد یک علامت «-» وجود دارد که با کلیک روی آن می‌توان ساب‌ایونت را موقّتاً پنهان کرد.

لزوم، از ساب‌ایونت‌ها برای سازمان‌دهی مجدّد استفاده کنید. تا آن موقع چرا از حالتی استفاده نکنید که خواندنش راحت‌تر است؟ البته خیلی وقت‌ها استفاده از ساب‌ایونت واقعاً ضروری است، مثل مواقعی که می‌خواهید کاندیشن‌های تکراری را حذف کنید یا موقعی که می‌خواهید از اکشن‌هایی استفاده کنید که در ایونتی دیگر استفاده کرده‌اید. در این صورت استفاده از ساب‌ایونت مشکلی ندارد؛ فقط زمانی استفاده از ساب‌ایونت مشکل دارد که اگر از آن استفاده نکنید بازهم نتیجه فرقی نداشته‌باشد، در این حالت است که ما توصیه می‌کنیم از آن استفاده نکنید.

یکی از رایج‌ترین مثال‌های استفاده‌ی بی‌جا از ساب‌ایونت، در حلقه‌های تودرتو است، مثل زیر:



این اشتباه از مقایسه‌ی حلقه‌های کانستراکت با زبان‌های برنامه‌نویسی سنّتی ناشی می‌شود، که معمولاً در آن‌ها برای نوشتن حلقه‌های تودرتو، حلقه‌ی داخلی را درون حلقه‌ی خارجی می‌نویسند. ولی در کانستراکت ۲ این کار لازم نیست، می‌توان حلقه‌های تودرتو را فقط در یک ایونت قرار داد، نتیجه هم فرقی نمی‌کند.



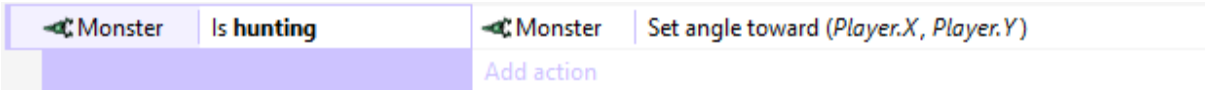
نکته‌ی کوچک دیگری نیز که باید مد نظر قرار داد این است که کانستراکت ۲ گاهی اوقات نمی‌تواند طوری با ساب‌ایونت‌ها رفتار کند که با ایونت‌های معمولی رفتار می‌کند. در مواردی نادر، عدم استفاده از ساب‌ایونت‌های اضافی می‌تواند عملکرد بازی را بهبود بخشد، مخصوصاً وقتی از حلقه‌هایی استفاده می‌کنید که بسیار هم تکرار می‌شوند. با این حال، باز هم نگرانی اصلی ما خوانایی ایونت‌شیت است.

حلقه‌ی For-each غیر لازم

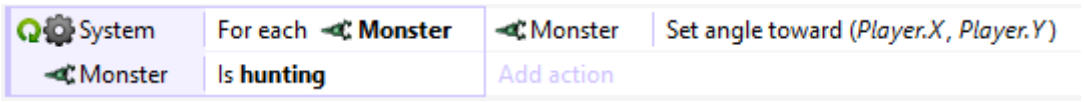
در بسیاری از مواقع حلقه‌ی For-each خیلی مفید است، امّا اغلب درجایی استفاده می‌شود که تغییری ایجاد نمی‌کند. نحوه‌ی عملکرد ایونت‌ها به این صورت است که ابتدا کاندیشن‌ها برای تک‌تک اینستنس‌ها بررسی می‌شوند و بعد اکشن‌ها ***برای هر اینستنسی***[1] که آن شرط برایش برقرار باشد اجرا می‌شوند. پس طبق این جمله، موتور کانستراکت ۲ از قبل یک حلقه‌ی for-each به صورت خودکار

[1] این عبارت، حلقه‌ی For each instance به معنی «برای هر اینستنس» را تداعی می‌کند.

دارد، تا به شما کمک کند ایونت‌ها آن طور که انتظار دارید عمل کنند. برای مثال ایونت زیر را در نظر بگیرید:



کانستراکت ۲ اینگونه ایونت بالا را اجرا می‌کند: ***برای هر*** هیولایی که دارد شکار می‌شود، زاویه‌اش را به سمت مکان پلیر تنظیم کن. با این حال گاهی می‌بینیم بعضی‌ها از ایونتی مشابه زیر استفاده می‌کنند:



در این مورد استفاده از حلقه‌ی For each زاید است. این نشان‌دهنده‌ی فهم نادرست از نحوه‌ی عملکرد ایونت‌های کانستراکت ۲ است. این کار اصلاً معنای ایونت را تغییر نمی‌دهد، کانستراکت ۲ به صورت خودکار این را انجام می‌دهد. این کار دو عیب دارد: اوّلاً این‌که استفاده از ایونت‌هایی که اثری در بازی نمی‌گذارند گیج‌کننده است، ولی نگرانی اصلی در این مورد عملکرد بازی است. حلقه‌ی for-each داخلی خود کانستراکت ۲ در سطح جاوااسکریپت به طور مستقیم اجرا می‌شود؛ حلقه‌های for-each که در ایونت‌شیت بازی وجود دارند نیز روی آن اجرا می‌شوند، که باعث می‌شود پردازش CPU بسیار بیشتر شود، در حالی‌که سازندگان کانستراکت ۲ به سختی کار کرده‌اند تا بازی را بهینه کنند، این کار زحمات‌شان را بر باد می دهد، پس در مواقع غیر ضروری اصلاً نباید از for-each استفاده کرد.

پس کی از حلقه‌ی for-each استفاده کنیم؟ فقط وقتی که حلقه‌ی خود انجین به تنهایی کافی نباشد، معمولاً زمانی که می‌خواهیم همه‌ی اکشن‌ها به ازای هر کدام از اینستنس‌ها به طور جداگانه اجرا شوند، نه فقط اکشن‌های مربوط به آن شیء. اغلب زمانی از این حلقه استفاده می‌کنیم که می‌خواهیم کاندیشن‌ها یا اکشن‌های سیستمی به ازای هرکدام از اینستنس‌ها یک‌بار اجرا شود (که در حالت عادی کلّاً یک‌بار اجرا می‌شوند، زیرا فقط یک اینستنس از شیء System وجود دارد). برای مثال، اگر در ایونت بالا یک اکشن سیستمی برای کم کردن مقدار یک متغیّر اضافه کنیم (Subtract from)، افزودن for-each به آن، عملکرد ایونت را تغییر می‌دهد. بدون این حلقه، هر وقت که ایونت اجرا شود، بدون توجّه به این‌که چند اینستنس از هیولا پیک شده‌است، فقط یک بار از مقدار متغیّر کم می‌شود؛ با وجود این حلقه، آن اکشن سیستمی به ازای هر کدام از هیولاها یک‌بار اجرا می‌شود، یعنی کم شدن مقدار متغیّر به تعداد هیولاهایی که پیک شده‌اند بستگی دارد. مطمئناً در این‌جا این حلقه مفید است، امّا مواردی که در آن‌ها استفاده از حلقه‌ی for-each مفید است خیلی کم تر از آن است که کاربران فکر می‌کنند. در اغلب موارد نیازی به استفاده از این حلقه نیست.

استفاده از dt در Every X seconds

کاربران زیادی آموزش «دلتاتایم و استقلال از نرخ فریم» را خوانده‌اند و از تکنیک‌هایش در پروژه‌هایشان استفاده می‌کنند (این آموزش انشاءالله به زودی ترجمه خواهد شد). امّا استفاده از dt (دلتاتایم) در کاندیشن Every X seconds ناشی از درک نادرست استقلال از نرخ فریم است.

ایونتی مثل Every 1 second به خودی خود مستقل از نرخ فریم است. این ایونت زمان را می‌سنجد، نه فریم‌ها را. در هر ثانیه یک‌بار اجرا می‌شود بدون توجّه به نرخ فریم. اگر نرخ فریم 10 FPS باشد در هر ثانیه یکبار اجرا می‌شود، اگر 60 FPS باشد هم در هر ثانیه یک‌بار اجرا می‌شود. این ایونت ربطی به نرخ فریم ندارد که بخواهد اصلاحش کنیم.

اتّفاقاً ایونتی مثل Every 60 * dt seconds وابسته به نرخ فریم است، برخلاف چیزی که می‌خواستید! در 10 FPS، مقدار dt برابر 0.1 است، بنابراین ایونت هر ۶ ثانیه یک‌بار اجرا می‌شود. در 60 FPS، مقدار dt حدود 0.016 است، بنابراین ایونت هر 0.96 ثانیه یک‌بار اجرا می‌شود. پس این مقدار وابسته به نرخ فریم است، و این چیزی بود که می‌خواستیم جلوی آن را بگیریم.

از طرف دیگر، اگر تنظیم کنیم که در هر تیک، شیئی یک پیکسل حرکت کند، این حرکت وابسته به نرخ فریم است، چون سرعت حرکت شیء به نرخ فریم بستگی دارد: در 30 FPS با سرعت ۳۰ پیکسل بر ثانیه و در 60 FPS با سرعت ۶۰ پیکسل بر ثانیه حرکت می‌کند. در این‌جا، می‌توان از dt استفاده کرد تا شیء با سرعت ثابت حرکت کند. dt زمان سپری شدن یک تیک است، بنابراین اگر شیئی را در هر تیک 60*dt پیکسل حرکت دهیم، آن شیء در هر ثانیه ۶۰ پیکسل حرکت می‌کند، خواه نرخ فریم 30 FPS باشد، خواه 60 FPS یا هر چیز دیگر. این هدف واقعی اکسپرشن dt است. از این اکسپرشن در ایونت‌هایی که خودشان مستقل از نرخ فریم هستند استفاده نکنید.

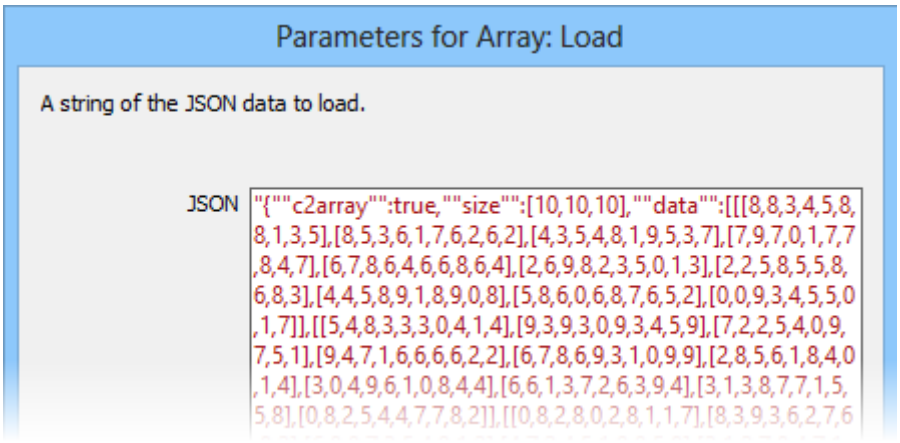
نوشتن اعداد خیلی کوچک در Every X seconds

اشتباه بعدی استفاده از مقادیر خیلی کوچک برای زمان در کاندیشن Every X seconds است، مثل Every 0.01 seconds. نرخ فریم بیشتر بازی‌ها 60 FPS است، که در این صورت زمان سپری شدن هر فریم حدود 16ms (شانزده هزارم ثانیه) است. هر کاندیشن یک‌بار در هر تیک بررسی می‌شود، پس حدّاکثر می‌تواند یک‌بار در هر تیک اجرا شود، بنابراین اگر زمانی را که در این ایونت می‌نویسید کمتر از 0.016 باشد، در هر تیک اجرا می‌شود، حتّی اگر نرخ فریم پایین بیاید، مثلاً در 10 FPS هر تیک حدود 0.1 ثانیه می‌شود، که باعث می‌شود باز هم ایونت شما در هر تیک اجرا شود، بنابراین، این کاندیشن

دقیقاً همان کار کاندیشن Every tick را انجام می‌دهد. و همان‌طور که گفته شد، خود Every tick نیز در اکثر مواقع اضافی است و می‌توان حذفش کرد. پس اگر کاندیشن Every X seconds دارای بازه‌ی زمانی کمتر از 0.016 ثانیه باشد، می‌توانید به راحتی آن را حذف کنید و هیچ فرقی در بازی‌تان نبینید.

Paste کردن اطّلاعات در یک اکسپرشن

گاهی اوقات اطّلاعات خیلی زیادی یا حتّی کلّ محتویات یک فایل در یک اکسپرشن پیست (Paste) می‌شود. مثل این:



به جای این کار از project files[1] استفاده کنید. اطّلاعات را در یک فایل متنی پیست کنید، آن را ذخیره کنید، از طریق نوار پروژه آن را وارد کنید، بعد از اکشن Request project file شیء AJAX برای بارگذاری آن استفاده کنید. حالا مگر Paste کردن اطّلاعات زیاد در یک اکسپرشن چه ایرادی دارد؟ این کار به چند دلیل بد است:

- همه‌ی اطّلاعات پیست شده باید ساختاری مثل ساختار اکسپرشن‌های کانستراکت داشته باشند تا از ارورهای ساختاری جلوگیری شود، مثلاً کاراکترهایی که درون دابل کوتیشن (" ") قرار می‌گیرند داخل یک رشته هستند، و اگر درون آن رشته بخواهیم از کوتیشن استفاده کنیم، باید به جای یک کوتیشن دوتا کوتیشن پشت سر هم بنویسیم تا با کوتیشن ابتدا و انتها اشتباه نشود. این کار برای فرمت‌هایی مثل JSON که معمولاً در آن‌ها از دابل کوتیشن استفاده می‌شود خسته‌کننده است.
- پیدا کردن، خواندن و تغییر دادن اطّلاعات در پنجره‌ی پارامترها سخت است.

[1] فایل‌های اضافه‌ای که از طریق پوشه‌ی Files در نوار پروژه می‌توان وارد کرد.

- برای نمایش همه‌ی اطّلاعات، ایونت‌شیت به شدّت دراز می‌شود، و برای گذشتن از ایونت مورد نظر باید خیلی پیمایش کنیم.
- بعد از خروجی گرفتن، همه‌ی اطّلاعات در فایل c2runtime.js ذخیره می‌شوند. قبل از این‌که حتّی بازی بخواهد در حالت لودینگ قرار گیرد این فایل باید به طور کامل دانلود و تجزیه‌شود و در حافظه قرار بگیرد. اگر اطّلاعات زیادی را در اکسپرشنی پیست کرده باشید حجم دانلود افزایش می‌یابد، و زمان ظاهرشدن لودینگ بازی افزایش می‌یابد، و همچنین استفاده از حافظه هم افزایش می‌یابد (مخصوصاً در موبایل).
- زمان باز کردن و ذخیره‌ی پروژه در ویرایشگر کانستراکت افزایش می‌یابد، زیرا اطّلاعات پیست شده در فایل‌های XML ذخیره می‌شوند (برای مشاهده‌ی این فایل‌ها پروژه‌تان را در حالت پوشه‌ای ذخیره کنید)، بنابراین هر بار که پروژه را باز می‌کنید باید تجزیه و تبدیل شوند.

Project file ها چنین مشکلاتی را ندارند:

- نیازی نیست محتویّات project file را برای سازگاری با ساختار اکسپرشن‌ها ویرایش کنید.
- پیدا کردن، خواندن و ویرایش project file خیلی راحت است.
- ایونت شیت زیاد دراز نمی‌شود، چون برای بارگذاری این فایل‌ها فقط به یک اکشن request شیء AJAX نیاز دارید.
- اطّلاعات، داخل فایل c2runtime.js نگهداری نمی‌شوند، بنابراین بازی سریع‌تر بارگذاری می‌شود و از حافظه هم کمتر استفاده می‌شود، و فقط موقعی که واقعاً لازم باشد دانلود می‌شوند.
- چون اطّلاعات خارج از پروژه هستند، باز شدن و ذخیره‌ی پروژه سریع‌تر است.

پس اگر می‌خواهید اطّلاعات زیادی را در ایونت‌ها پیست کنید، به شدّت توصیه می‌کنیم به جای آن از project file استفاده کنید.

توقّع دقیق بودن محاسبات ریاضی

تمام کامپیوترهای مدرن اعداد اعشاری را در قالبی مثل 0.5 نگهداری می‌کنند. چون حافظه و قدرت پردازش کامپیوتر بی‌نهایت نیست، کامپیوتر مجبور می‌شود کمی اعداد را گرد کند. این به نوبه‌ی خود می‌تواند باعث شود جوابی که کامپیوتر به دست می‌آورد کمی دور از جواب حقیقی باشد، مثلاً در جایی که توقّع دارید کامپیوتر به شما عدد 1 را نشان دهد عدد 0.999999 را نمایش می‌دهد.

حالا توضیح می‌دهیم این مشکل چرا پیش می‌آید. تقسیم عدد 1 بر 3 را در نظر بگیرید. جواب می‌شود 0.3333333... که همین طور تا ابد 3 تکرار می‌شود. امّا کامپیوتر حافظه‌ی محدودی دارد و نمی‌تواند بی‌نهایت عدد مثل این بی‌نهایت 3 را در حافظه‌اش نگه دارد. بنابراین فقط تعداد محدودی از اعشار را ذخیره می‌کند و بعد از آن، محاسبه را متوقّف می‌کند. مثلاً فقط تا ۶ رقم اعشار را نگه می‌دارد و جواب می‌شود 0.333333 . این از لحاظ ریاضی غلط است، ولی کامپیوتر نمی‌تواند جواب درست را نگه دارد. حالا فرض کنید این جواب را مجدداً در ۳ ضرب کنیم. شما انتظار دارید سه ضرب‌در یک‌سوّم برابر ۱ شود، ولی در حقیقت عدد 0.999999 را دریافت می‌کنید که خیلی به جواب اصلی نزدیک است، ولی دقیق نیست. اگر بازی شما بخواهد که جواب **دقیقاً** مساوی ۱ شود، آنگونه که انتظار دارید عمل نمی‌کند. معمولاً می‌توانید از طریق دیباگر (debugger) جوابی را که واقعاً محاسبه می‌شود بررسی کنید.

این نوع مشکل در محاسبات اعشاری در اغلب نرم‌افزارها وجود دارد. راه فراری نیست! این حقیقت نحوه‌ی عملکرد کامپیوترهاست. مثال ما در بالا در مبنای ۱۰ بود، درحالی‌که کامپیوتر در مبنای ۲ کار می‌کند، و دوره‌ی گردش رقم‌های اعشار در مبنای ۲ با مبنای ۱۰ فرق می‌کند. یعنی 0.1 در مبنای ۱۰ یک عدد اعشاری مختوم است و دوره‌ی گردش ندارد، ولی همین عدد وقتی به مبنای ۲ می‌رود دارای دوره‌ی گردش می‌شود. یعنی این مشکل حتّی در مواقعی که انتظارش را هم ندارید می‌تواند اتّفاق بیفتد، حتّی اگر فکر کنید که دارید با اعداد دقیقی محسابه می‌کنید. این مشکل مخصوصاً در محلّ اشیاء اثر می‌گذارد، چون حرکت یک شیء در زاویه‌ای خاص معمولاً نیاز به محاسبات سینوس و کسینوسی دارد، که معمولاً جواب بسیار نزدیکی به جواب اصلی به دست می‌آید، ولی دقیق نیست.

راهکاری که می‌توانید از آن در هر نرم‌افزار و فریم‌ورکی برای حلّ این مشکل استفاده کنید این است که مقداری تلرانس[1] را مجاز اعلام کنید. توقّع نداشته‌باشید ایونتی مثل **Sprite.X = 100** همیشه درست کار کند. به جای آن از چیزی مثل **abs(Sprite.X – 100) < 0.01**[2] استفاده کنید. این کار مقایسه‌ی شما را درست می‌کند، زیرا اگر نتیجه تا ۰,۱ از ۱۰۰ فاصله داشت، باز هم ایونت اجرا می‌شود. پس اگر مثلاً عدد ما ۹۹,۹۹۹۹۹۹ شد باز هم همان طور که انتظار داریم، ایونت اجرا می‌شود.

[1] تلرانس: به خطای مجاز تلرانس گفته می‌شود، مثلاً می‌گوییم اگر جواب به دست آمده تا 0.2 با جواب اصلی فرق داشت عیبی ندارد (در این حالت تلرانس ما 0.2 است).

[2] طبق اصول ریاضی، قدر مطلق تفاضل دو عدد، فاصله‌ی آن دو از هم را نشان می‌دهد. پس در این‌جا گفته‌ایم اگر فاصله‌ی بین Sprite.X و 100 کمتر از 0.01 باشد اکشن‌ها را اجرا کن (در این مثال تلرانس 0.01 است)

Pathfinding افراطی

رفتار pathfinding می‌تواند بسیار مفید باشد، ولی یکی از رفتارهایی است که به شدّت از CPU کار می‌کشد.[1] یافتن یک مسیر، عملیّات بسیار پیچیده‌ای است که اغلب در آن هزاران راه مختلف برای پیدا کردن یک مسیر خوب بررسی می‌شود، و هر بار که از اکشن Find path استفاده می‌کنید این عملیّات برای هر کدام از اینستنس‌ها تکرار می‌شود! خوشبختانه عملیّات جستجوی مسیر به طور موازی با اجرای خود بازی انجام می‌شود، این باعث می‌شود که در حین این عملیّات، بازی هنگ نکند. ولی شما می‌توانید یافتن مسیر را بسیار سخت کنید: بعضی از کابران بی‌تجربه، اندازه‌ی سلّول‌ها (cell size) را فقط چند پیکسل، یا حتّی ۱ پیکسل قرار می‌دهند. این کار مثل این است که در دنیای واقعی بخواهیم برای یافتن یک مسیر، اتم به اتم، راه را بررسی کنیم تا یک مسیر مناسب برای رسیدن به مقصد پیدا کنیم، ولی متأسفانه قدرت پردازش کامپیوتر شما نامحدود نیست. تلاش برای انجام این کار باعث می‌شود که به طرز غیر منطقی استفاده از CPU خیلی زیاد شود، و پیدا شدن مسیر خیلی به طول بینجامد، یا حتّی متوقّف شود، مخصوصاً در موبایل.

راه حل ساده است: تا حدّ امکان اندازه‌ی سلّول‌ها را بزرگ قرار دهید. وقتی اندازه ی سلّول‌ها بزرگ باشد، گام‌های کمتری برای پیدا کردن مسیر لازم است، در نتیجه مسیر سریع‌تر پیدا می‌شود. به همین دلیل به طور پیش‌فرض از یک شبکه‌ی ۳۰×۳۰ پیکسلی استفاده می‌شود، این کار برای بهینه‌سازی بازی است، برای این است که مسیر، سریع‌تر از زمانی پیدا شود که بخواهیم نگران تک‌تک پیکسل‌ها باشیم.

بازسازی نقشه‌ی موانع (obstacle map) نیز به خاطر این‌که تمام لیوت را در بر دارد برای CPU سخت است، و تا حدّ امکان باید از آن کم استفاده کنید. با کوچک کردن بیش از حدّ اندازه‌ی سلول‌ها، تلاش برای یافتن مسیر در هر تیک، یا بعد از پیدا شدن مسیر قبلی، بار شدیدی به CPU تحمیل می‌شود که شاید نتواند از پسش بر آید. سعی کنید چنین کارهایی را انجام ندهید.

حرف آخر

امیدوارم این مقاله به شما کمک کرده‌باشد تا این اشتباهات را در سیستم ایونت مرتکب نشوید. اشتباه رایج دیگری که وجود دارد این است که عده‌ای فکر می‌کنند حافظه‌ی کامپیوتر نامحدود است، انشاءالله در این مورد در آینده بیشتر توضیح می‌دهیم.

مجتبی قاسم‌زاده تهرانی

دی ۱۳۹۴

[1] رفتار دیگری که خیلی از CPU کار می‌کشد، فیزیک است.