وابسنگی های ساخت نه تنها بر تغییر به خود دارایی‌ها تأثیر دارند بلکه همچنین به تغییر به فورمت های داده‌ای هم وابسته هستند. اگر برای مثال فورمت تمام فایل‌های استفاده شده برای ذخیره ی مش های مثلث تغییر کند، تمام مش ها در تمام بازی ممکن است نیاز به دوباره استخراج و دوباره سازی داشته باشند. بعضی موتورهای بازی فورمت های داده‌ای را مورد استفاده قرار می‌دهند که در برابر تغییرات نسخه مقاوم هستند. برای مثال یک دارایی ممکن است شامل یک شماره ی نسخه باشد و موتور بازی ممکن است شامل کدی باشد که می‌داند چگونه از دارایی‌های قدیمی استفاده کند و آن‌ها را بارگزاری کند. مشکلات چنان سیاستی این است که فایل دارایی و کد موتور تمایل دارند که بسیار حجیم شوند. هنگامی که تغییر فورمت های داده‌ای بسیار نادر است، ممکن است بهتر باشد که تنها گلوله را گاز گرفت و تمام فایل‌ها را هنگامی که تغییر فورمت روی می‌دهد، دوباره پردازش کرد.

خط لوله های ساده ی دارایی نیاز به مجموعه‌ای از قوانین دارند که وابستگی‌های درونی را بین دارایی‌ها توضیح می‌دهد و نوعی از ابزار ساخت که می‌تواند از این اطلاععات برای اطمینان از اینکه یک دارایی خاص، به ترتیب مناسب ساخته شده، در هنگام ویرایش یک دارایی منبع استفاده کند. تعدادی تیم های بازی سیستم ساخت خودشان را می سازند. سایرین از یک ابزار تثبیت شده استفاده می‌کنند مانند make. هر راه حلی که انتخاتب شود،‌تیم باید با سیستم وابستگی ساختشان با احتیاط کامل رفتار کنند. اگر شما اینگونه نکنید، تغییرات به کد منبع دارایی ممکن است باعث نشود که دارایی مناسب دوباره سازی شود. نتیجه می‌تواند با دارایی‌های بازی متناقض باشد، که خود ممکن است باعث موارد غیر طبیعی بصری شود و یا حتی باعث کرش موتور شود. به تجربه ی شخصی، من ساعات بیشماری را دیده‌ام که صرف پیدا کردن مشکلاتی شده است که می‌توانست پیشگیری شود اگر وابستگی‌های درونی دارایی‌ها به خوبی مشخص می‌شد و سیستم ساخت برای استفاده ی آن‌ها به صورت قابل اعتمادی پیاده‌سازی می شد.

6.2.2 مدیریت منابع در هنگام اجرا

اجازه دهید که توجهمان را به معطوف کنیم که چگونه دارایی‌ها در پایگاه داده ی منابع ما بارگزاری می شوند، مدیریت می‌شوند و از موتور خارج می‌شوند در هنگام اجرا.

6.2.2.1

یک مدیریت منابع موتور بازی در هنگام اجرا مجموعه ی وسیعی از وظایف را بر عهده دارد، که همه آن‌ها مرتبط با وظیفه ی اصلی آن یعنی بارگزاری منابع به درون حافظه است:

* اطمینان از اینکه تنها یک کپی از هر منبع واحد در هر زمان در حافظه موجود است.
* مدیریت زمان حیات هر منبع
* بارگزاری منابع مورد نیاز و خارج کردن منابعی که دیگر مورد نیاز نیستند.
* کنترل بارگزاری منابع مرکب. یک منبع مرکب یک منبع است که شامل سایر منابع است. برای مثال یک مدل سه بعدی یک منبع مرکب است که شامل یک مش، یک یا بیشتر ماده، یک یا بیشتر بافت و به صورت دلخواه یک اسکلت و چندین انیمیشن اسکلت است.
* تأمین تمامیت ارجاعی. این شامل تمامیت ارجاعی درونی است (رفرنس میانی درون یک تک منبع) و همچنین تمامیت ارجاعی خارجی (رفرنس میانی بین منابع). برای مثال یک مدل به مش و اسکلت خود اشاره می کند؛ یک مش به مواد خود اشاره می کند؛ که در حقیقت خود به منابع بافتی اشاره می کند؛ انیمیشن به یک اسکلت اشاره می‌کند که خود در نهایت آن‌ها را به یک یا چند مدل مرتبط می کند. هنگام بازیابی یک منبع مرکب، مدیر منبع باید اطمینان حاصل کند که تمام زیر منابع ضروری بارگزاری شده‌اند و باید تمام ارجاعات میانی را به خوبی قرار دهد.
* استفاده از حافظه ی منابع بازگزاری شده را مدیریت می‌کند و اطمینان پیدا می‌کند که منابع در جاهای درست خود در حافظه ذخیره شده اند.
* اجازه می‌دهد که پردازش سفارشی بر روی یک منبع بعد از اینکه بارگزاری شد اجرا شود، به یک شکل وابسته به نوع منبع. این فرایند گاهی اوقات به عنوان وارد شدن یا شروع بارگزاری منبع شناخته می شود.
* معمولاً (اما نه همیشه) یک فصل مشترک یکی شده را ایجاد می‌کند که از طریق آن مجموعه وسیعی از نوع منابع می‌توانند مدیریت شوند. به صورت ایده‌آل یک مدیر منبع همچنین به سادگی قابل توسعه است، تا بتواند انواع جدید منابع را در همان حین که توسط تیم توسعه‌دهنده ی بازی مورد نیاز است کنترل کند.
* کنترل جریان سازی (مثلاً بارگزاری آسنکرون منابع)، اگر موتور این قابلیت را پشتیبانی کند.

6.2.2.2 فایل منبع و سازمان دهی دایرکتوری

در بعضی موتورهای بازی (مخصوصاً موتورهای PC)، هر منبع واحد به صورت یک فایل شل بر روی دیسک مدیریت می شود. این فایل‌ها مخصوصاً درون یک درخت از دایرکتوری ها وارد شده‌اند که سازمان دهی درونی آن از پیش طراحی شده است تا کار افراد سازنده ی دارایی‌ها را آسان کند؛ موتور مخصوصاً اهمیت نمی‌دهد کجا فایل‌های منابع درون درخت منابع قرار دارند. اینجا یک دایرکتوری معمولی درختی برای یک بازی فرضی به نام Space Evaders وجود دارد:

SpaceEvaders دایرکتوری ریشه برای کل بازی

ًResources ریشه ی تمام منابع

NPC مدل و انیمیشن کاراکتر های غیر قابل بازی

Pirate مدل و انیمیشن ها برای دزدان دریایی

Marine مدل و انیمیشن ها برای دریانوردان

Player مدل و انیمیشن های کاراکترهای بازیباز ها

Weapons مدل و انیمشن ها برای سلاح ها

Pistol مدل و انیمشن ها برای سلاح دستی

Rifle مدل و انیمیش ها برای سلاح

BFG مدل و انیمیشن ها برای اسلحه ی بزرگ

….

Levels هندسه ی پشت صحنه و ظاهر مراحل

Level1 منابع مرحله ی اول

Level2 منابع مرحله ی دوم

…

Objects اشیاء سه بعدی متفرقه

Crate ظرف همه جا حاضر قابل شکستن

Barrel بشکه ی همه جا حاضر قابل انفجار

سایر موتورها چندین منبع را در یک تک فایل، بسته بندی می کنند مانند یک آرشیو zip یا فایل ترکیبی از نوعی دیگر (احتمالا از یک نوع سفارشی). فایده ی اولیه ی این روش این است که زمان بارگزاری را کاهش می دهد. هنگام بارگزاری داده از یک فایل، بزرگترین سه تا هزینه عبارتند از زمان جستجو (مثلا انتقال سر خواندن یه جای درست بر روی وسیله ی فیزیکی)، زمان مورد نیاز برای باز کردن هر تک فایل و زمان برای خواندن داده از فایل به درون حافظه. از اینها، زمان جستجو و زمان باز کردن فایل می تواند در مقایسه با بسیاری از عملیات سیستم بالا باشد. هنگامی که یک فایل یکتای بزرگ مورد استفاده قرار می گیرد، تمام این هزینه ها حداقل می شوند. یک تک فایل می تواند به صورت دنباله ای سازماندهی شود تا زمان جستجو را به حداقل برساند. و با تنها یک فایل برای باز کردن، هزینه ی باز کردن تک فایل ها حذف می شود.

دیسک های حالت جامد از مشکلات مربوط به زمان جستجو که مشکلات زیادی را برای وسایل چرخشی مانند DVD، دیسک بلوری و دسیک های سخت ایجاد کرده است، رنج نمی برند. هرچند هیچ کنسول بازی ای شامل یک درایو حالت جامد به عنوان وسیله ی ثابت اصلی ذخیره نیست (نه حتی PS4 و Xbox one). بنابراین طراحی الگوهای ورودی/خروجی بازی شما برای به حداقل رسانی زمان جستجو احتمالا یک ضرورت برای زمان های آینده است.

مدیریت کننده ی منابع موتور رندر اوگر به منابع اجازه می دهد تا به عنوان فایل های شل بر روی دیسک و یا یک فایل مجازی درون یک آرشیو بزرگ ZIP وجود داشته باشند. فایده ی اولیه ی فورمت ZIP این ها هستند:

1. یک فورمت باز است. کتابخانه های zlib و zziplib استفاده شده برای خواندن و نوشتن آرشیو های ZIP آزادانه در دسترس هستند. SDK ی zlib کاملا مجانی است، در حالی که sdk ی ZZIPLIB تحت مجوز Lesser Gnu Public License است.
2. فایل مجازی درون یک آرشیو zip یه یاد دارد که مسیرهای نسبی آن چه هستند. این بدین معناست که یک آرشیو zip به نظر یک سیستم فایل خام است برای بیشتر استفاده ها و اهداف. مدیر منابع اوگر تمام منابع را با استفاده از رشته هایی که به نظر مسیرهای سیستم فایل هستند تشخیص می دهد. هرچند این مسیرها گاهی اوقات فایل های مجازی درون یک آرشیو zip را به جای فایل های شل درون دیسک تشخیص می دهد و یک برنامه نویس بازی نیاز ندارد که از تفاوت ها در بیشتر موقعیت ها آگاه باشد.
3. آرشیو zip ممکن است فشرده باشد. این مقدار فضای دیسک پرشده توسط منابع را کاهش می دهد. اما مهم تر از آن بازهم زمان های بارگزاری را کاهش می دهد زیرا داده های کمتری باید بارگزاری شوند درون حافظه از یک دیسک مشخص. این مخصوصا به ما کمک می کند هنگام خواندن داده از یک رام DVD یا رام Bluray، زیرا همان طور که نرخ انتقال داده از این ابزارها بسیار کندتر از یک درایو دیسک سخت است. بنابراین هزینه ی تجزیه ی داده بعد از بارگزاری به درون حافظه معمولا بیشتر است از آفستی که زمان صرفه جویی شده از بارگزاری داده های کمتر به درون دیسک برای ما ایجاد می کند.
4. آرشیو های Zip ماژولار هستند. منابع می توانند با همدیگر به درون یک فایل ZIP گروه بندی شوند و به عنوان یک واحد مدیریت شوند. یک کاربر بسیار عالی از این ایده در ایجاد محلی سازی است. تمام این دارایی ها که نیاز به محلی سازی دارند (مانند کلیپ های صوتی شامل دیالوگ و بافت که شلمل کلمات یا سمبل های وابسته به موقعیت هستند) می توانند درون یک فایل تک ZIP قرار گیرند و سپس نسخه های مختلفی از این فایل های زیپ می توانند ایجاد شوند یکی برای هر یک از زبان ها یا مناطق. برای اجرای بازی برای یک منطقه ی خاص، موتور به آسانی نسخه ی منطبق از آرشیو زیپ را بارگزاری می کند

آنریل 3 از یک روش مشابه استفاده می کند با تعدادی تفاوت مهم. در آنریل تمام منابع باید درون مجموعه های بزرگ وارد شوند که به عنوان بسته شناخته می شوند (همان pack file). هیچ دیسک فایل شلی اجازه داده نمی شود. فورمت یک فایل بسته نسبی است. ویرایشگر موتور بازی آنریل، انریل اد به توسعه دهندگان اجازه می دهد که بسته های را ایجاد و مدیریت کنند و همچنین منابعی که درون آنهاست.

6.2.2.3 فورمت های فایل منبع

هر نوع از فایل منبع به صورت پتانسل فورمت متفاوتی دارد. برای نمونه یک فایل مش همیشه به صورت یک فورمت متفاوت از یک بیت مپ بافت ذخیره می شود. انواعی از دارایی ها در فورمت های استاندارد شده و باز ذخیره می شوند. برای نمونه بافت ها معمولا به عنوان فایل های تارگا ذخیره می شوند، فایل های گرافیک شبکه ی قابل حمل، فایل فورمت فایل عکس تگ شده، فایل گروه کارشناس تصویری مشترک یا فایل بیت مپ ویندوز یا در یک فورمت فشرده و استاندارد شده مانند خانواده ی فشرده بافت متعلق به دارکت اکس اس 3 از فورمت ها. به صورت مشابه داده های مش سه بعدی معمولا از یک ابزار مدلسازی سه بعدی مانند مایا یا لایت ویو به صورت یک فورمت استاندارد شده مانند obj یا collada برای استفاده توسط موتور بازی استخراج می شوند.

گاهی اوقات همان طور که یک تک فورمت فایل می تواند توسط موتور بازی مورد استفاده قرار گیرد.

گاهی اوقات یک تک فایل فورمت می تواند برای جای دادن بسیاری انواع مختلف دارایی استفاده شود. برای مثال، اس دی کی مربوط به Granny توسط Rad Game Tools یک فورمت فایل قابل انعطاف را پیاده سازی می کند که می تواند برای ذخیره ی داده های مش سه بعدی، سلسله مراتب اسکلتی و داده های انیمیشن اسکلتی مورد استفاده قرار گیرد. (در حقیقت فورمت فایل گرانی می تواند به آسانی برای ذخیره تقریبا هر نوع از داده های قابل تصور بازطراحی شوند.)

بسیاری از برنامه نویسان موتورهای بازی فایل فورمت خودشان را به دلایل مختلفی مورد استفاده قرار می دهند. این می تواند الزامی باشد اگر هیچ فورمت استانداردی ارایه دهنده ی تمام اطلاعات مورد نیاز توسط موتور نباشد. همچنین بسیاری از موتورهای بازی تلاش می کنند تا تا جایی که ممکن پردازش های آفلاین انجام دهند تا اندازه ی زمان مورد نیاز برای بارگزاری و پردازش داده های منابع را در زمان اجرا را به حداقل برسانند. اگر داده های نیاز داشته باشند که با یک شکل خاصی در حافظه منطبق باشند، برای مثال یک فورمت خام دودویی ممکن است برای آن انتخاب شده باشد تا اینکه داده ها بتوانند با استفاده از یک ابزار آفلاین پخش شوند (به جای اینکه سعی کنیم تا آنها را در زمان اجرا بعد از اینکه منابع بارگزاری شدند سعی در حذف آنها داشته باشیم)

6.2.2.4 GUID های منابع

هر منبع در یک بازی باید تعدادی مشخص کننده های جهانی یکتا (GUID) داشته باشد. عمومی ترین انتخاب جی یو آیدی ها مسیر فایل سیستم یک فایل منبع است (به عنوان یک رشته یا یک مش 32 بیتی ذخیره شده است). این نوع از جی یو آیدی ها بصری هستند، زیرا به وضوح هر منبع را به یک فایل فیزیکی بر روی دیسک وصل می کنند. و تضمین شده است که در تمام بازی یکتا باشد، زیرا سیستم عامل خودش تضمین کرده است که هیچ دو فایلی یک آدرس نداشته باشند.

هرچند یک آدرس سیستم عامل به هیچ صورتی تنها انتخاب برای یک جی یو آیدی منبع نیست. بعضی از موتورها از یک نوع کمتر بصری از جی یو آیدی استفاده می کنند مانند یک کد هش 128 بیتی، شاید توسط یک ابزار تخصیص داده شده باشند که یکتایی را تضمین می کند. در سایر موتورها، استفاده از یک آدرس سیتم فایل به عنوان یک مشخص کننده ی منبع عملی نیست. برای نمونه موتور آنریل 3 بسیاریاز منابع را در یک تک فایل بزرگ که به عنوان یک بسته شناخته می شود دخیره می کند، بنابراین مسیر به فایل بسته به صورت بکتا هیچ یک از منابع را مشخص نمی کند. برای حل این مشکل، یک بسته ی فایل آنریل به یک سلسله مراتب پوشه ای سازمان دهی می شود که شامل منابع یکتاست. آنریل به هر تک منیع درون یک بسته یک نام یکتا می دهد که بسیار شبیه یک مسیر فایل سیستم است. بنابراین در آنریل، یک جی یو آیدی منبع با اتصال نام (یکتا) بسته فایل با مسیر درون بسته از منبع مورد نظر ایجاد می شود. برای مثال جی یو آیدی منبع بازی Gears of war در Locust\_Boomer.Physical-Materials.LocustBoomerLeather نشان دهنده ی یک ماده است که به عنوان Locust-BoomerLeather درون پوشه ی PhysicalMaterials از بسته فایل Locust\_Boomer قرار دارد.

6.2.25 ثبت کننده ی منبع

برای اطمینان از اینکه تنها یک کئی از هر تک منبع به درون حافظه بارگزاری می شود در هر زمان، بیشتر مدیران منابع نوعی از ثبات منابع بارگزاری شده را تامین می کنند. ساده ترین ئیاده سازی یک دیکشنری است – یعنی یک مجموعه از جفت های کلید-ارزش. کلیدها شامل آیدی های تکی از منابع هستند در حالی که ارزش ها معمولا اشاره گرهایی به منابع درون حافظه هستند.

هر هنگام که یک منیع به درون حافظه بارگزاری می شود، یک ورودی برای آن، به دیکشنری ثبت کننده ی منبع اضافه می شود با استفاده از جی یو آیدی آن به عنوان کلید. هر هنگام که یک منبع خارج می شود، ورودی ثباتی آن حذف می شود. هنگامی که یک منبع درخواست می شود توسط بازی، مدیر منبع، منبع را با استفاده از جی یو آیدی آن درون ثبات منابع ئیدا می کند. اگر منبع ئیدا شد، یک اشاره گر به آن به آسانی برگردانده می شود. اگر منبع را نتوان ئیدا کرد، می تواند یا به صورت اتوماتیک بارگزاری شود و یا یک کد خطا بازگردانده شود.

در اولین برخورد، ممکن است بسیار بصری به نظر برسد که به صورت اتوماتیک یک منبع خواسته شده را بارگزاری کرد اگر نتوان آن را ئیدا کرد درون ثبات منابع. و در حقیقت تعدادی از موتورهای بازی این کار را انجام می دهند. هرچند تعدادی مشکل جدی وجود دارد با این روش. بارگزاری یک منبع یک عملیات کند است زیرا شامل بارگزاری و بازکردن یک فایل در دیسک می شود، خواندن یک تعداد زیاد از داده به درون حافظه (از یک ابزرا کند مانند یک دی وی دی خوان)، و همچنین احتمالا اجرای شورع ئس از بارگزاری داده ی منبع هنگامی که بارگزاری شد. اگر درخواست در طول گیم پلی فعال بیاید، زمان مورد نیاز برای آن برای بارگزاری منبع ممکن است باعث یک وقفه ی قابل دیدن در نرخ فریم بازی شود و یا حتی یک توقف چندثانیه ای. برای این دلیل موتورها تمایل دارند که یکی از دو روش را برگزینند:

1. بارگزاری منبع ممکن است کاملا لغو شود در صورل گیم ئلی فعال. در این مدل تمام منابع برای یک مرحله ی بازی به صورت کلی قبل از گیم ئلی بارگزاری می شوند که معمولا بازیباز یک صفحه ی بارگزاری را می بیند و نوعی از نوار ئیشرفت را مشاهده می کند.
2. بارگزاری منبع ممکن است به صورت آسنکرون انجام شود.(یعنی داده ممکن این جریان ئیدا کند). در این مدل در حالی که بازیکن درگیر مرحله ی x است، منابع مورد نیاز برای مرحله ی y بارگزاری می شوند در ئشت صحنه. این روش ترجیح دارد زیرا به بازیکن یک تجریه ب بازی بدون صفحه ی انتظار می دهد. هرچند به صورت قابل توجهی ئیاده سازی دشوار تری دارد.

6.2.2.6 زمان حیات منبع

زمان حیات یک منبع به عنوان زمان بین هنگامی که اولین بار به درون حافظه بارگزاری می شود و هنگامی که حافظه ی آن برای سایر اهداف گرفته می شود تعریف می شود. یکی از کارهای مدیر منابع این است که زمان حیات منبع را مدیریت کند—چه به صورت اتوماتیک و یا با تامین توابع API ضروری برای بازی، بنابراین می تواند به صورت دستی زمان حیات منبع را مدیریت کند.

هر منبع نیازمندی های زمان حیات خود را دارد.

* بعضی منابع باید هنگامی که بازی بار اول اجرا می‌شود بارگزاری شوند و باید سراسر طول بازی در حافظه بمانند. به این ترتیب طول عمرشان عملاً بی نهایت است. این‌ها چیزهایی هستند که به عنوان منابع جهانی یا دارایی‌های جهانی شناخته می شوند. معمولاً مثالهایی از این‌ها شامل مش کاراکتر بازیکنُ مواد، بافت و انیمشن های اصلی، بافت و فونت‌های استفاده شده برای هد آپ دیسپلی و منابع برای تمام اسلحه های استاندارد استفاده شد هدر تمام بازی هستند (و نمی‌توانند در هنگام نیاز فوراً بارگزاری شوند) و باید به عنوان یک منبع جهانی با آن‌ها رفتار شود.
* سایر منابع یک زمان حیات دارند که مطابق با یک مرحله ی خاص از بازی است. این منابع باید در حافظه بمانند از زمانی که مرحله بار اول توسط بازیکن دیده می‌شود و باید هنگامی که بازیکن کاملاً مرحله را به پایان برد از حافظه خارج شوند.
* بعضی منابع ممکن است یک زمان حیات داشته باشند که کمتر از طول مرحله‌ای است که در آن هستند. برای نمونه انیمیشن و کلیپ صوتی که سینماتیک های درون بازی را می‌سازند (یک مینی فیلم که داستان را به جلو می‌برد و یا به بازیکن اطلاعات مهمی می دهد) ممکن است پیش از دیدن فیلم توسط بازیکن بارگزاری شوند و سپس هنگامی که سینماتیک اجرا شد از حافظه خارج شوند.
* بعضی منابع مانند موزیک پشت صحنه، صداهای محایط یا فیلم‌های تمام صفحه به صورت زنده به جریان در می‌آیند در همان حین که اجرا می شوند. زمان حیات این نوع از منابع برای تعیین کردن دشوار است زیرا هر بایت تنها برای جز کوچکی از ثانیه در حافظه باقی می‌ماند اما کل قطعه ی صدای موزیک این‌طور به نظر می‌رسد که برای مدت زمان طولانی باقی می ماند. چنان دارایی‌هایی معمولاً در بخش‌های با اندازه‌ای که متناسب با نیازمندی های سخت‌افزار باشد در حافظه بارگزاری می شوند. برای نمونه یک کلیپ موسیقی ممکن است در بخش‌های ۴ KB خوانده شود زیرا ممکن است بافر سایز استفاده شده توسط سیستم صوتی سطح پایین باشد. تنها دو بخش در هر لحظه‌ای در حافظه وجود دارند –بخشی که اکنون در حال پخش است و بخشی که فوراً آن را دنبال می‌کند که در حال بارگزاری به حافظه است.

سؤال این است که چه هنگام منبع معمولاً بارگزاری می‌شود که به راحتی پاسخ داده می‌شود بر اساس دانش اینکه چه هنگام دارایی برای بار اول توسط بازیکن دیده می شود. هرچند سؤال اینکه چه زمان منبع را از حافظه خارج و حافظه را پس گیریم به همین سادگی پاسخ داده نمی شود. مشکل این است که بسیاری از منابع بین چندین مزحله به اشتراک گذاشته می شوند. ما نمی‌خواهیم منبعی را خارج کنیم هنگام پایان یافتن مرحله ی x تا اینکه فوراً دوباره آن را بارگزاری کنیم زیرا مرحله ی y به آن نیاز دارد.

تنها راه حل برای این مشکل استفاده شمردن ارجاع منبع است. هنگامی که یک مرحله بازی جدید باید بارگزاری شود، لیست تمام منابع استفاده شده توسط آن مرحله پیمایش می‌شود و شمارش مرجع برای هر منبع توسط یک افزایش می‌یابد (اما هنوز بارگزاری نشده اند). بعد ما منابع هر مورحله خواسته نشد هرا پیمایش می‌کنیم و شمارنده ی مرجعشان را یکی کم می‌کنیم هر منبعی که شمارنده ی مرجعش به صفر برسد از حافظه خارج می شود.در نهایت ما از لیست تمام منابع که مراجعشان از صفر به یک رسیده است می‌گذریم و آن دارایی‌ها را به درون حافظه بارگزاری می کنیم.

برای مثال فرض کنید که مرحله ی x از منابع A،B و C استفاده می‌کند و آن مرحله Y از منابع B، C،D و E استفاده می‌کند (B و C بین هر دو مرحله به اشتراک گذاشته شده اند). جدول ۶.۲ نشان می‌دهد که شمارش مرجع این پنج منبع چگونه در حینی که بازیکن از مرحله ی X به مرحله ی Y می‌رود انجام می شود. در این جدول شمارنده های مرجع به صورت بولد شده نشان داده شده‌اند تا نمایش دهنده ی منابع منطبقی باشند که تا کنون در حافظه موجودند در حالی که یک پیش‌زمینه خاکستری به معنی این است که منبع در حافظه موجود نیست. یک شمارش ارجاع در پرانتز نشان دهنده ی است است که منبع منطبق داده در حال بارگزاری یا خارج شدن است.

۶.۲.۲.۷ مدیریت حافظه برای منابع

مدیریت حافظه با مدیریت منابع ارتباط تنگاتنگی دارند زیرا ما باید به صورت غیرقابل اجتنابی تصمیم بگیریم که کجا منابع باید در هنگامی که در حافظه بارگزاری شدند قرار بگیرند. سرنوشت هر منبع همیشه یکسان نیست. برای نمونه انواع خاصی از منایع باید در رم ویدیویی (یا در پلی استیشن ۴ در یک بلوک حافظه که برای دسترسی با باس پرسرعت گارلیک مسیرگذاری شده است) قرار گیرند. مثالهای معمولی شامل بافت، بافر راس، بافر اندیس و کد سایه زن است. بیشتر سایر منابع می‌توانند در رم اصلی قرار گیرند اما انواع مختلفی از منابع ممکن است نیاز به قرار گرفتن در بازه های مختلفی از آدرس را داشته باشند. برای نمونه یک منبع که بارگزاری شده و در طول کل بازی باقی می‌ماند (منابع جهانی) ممکن است به داخلی یکی از مناطق حافظه بارگزاری شود در حالی که منابع که مکرراً بارگزاری و خارج می‌شوند ممکن است جای دیگری بروند.

طراحی یک زیر سیستم تخصیص حافظه ی موتور بازی معمولاً ارتباط زیادی با مدیر منبع آن دارد. بعضی اوقات ما مدیر منبع را به گونه اس طراحی می‌کنیم تا از مزایای نوع تخصیصی که داریم لستفاده کند و برعکس – ما ممکن است تخصیص دهنده ی حافظه مان را برای تأمین نیازهای مدیر منبع طراحی کنیم.

ما در بخش 5.2.1.4 دیدم که یکی از مسایل اصلی که هر مدیر منبع را تهدید می‌کند نباز برای جلوگیری از پارگی حافظه است در هنگامی که منابع بارگزاری و خارج می شوند. ما تعدادی از راه حل‌های معمول را برای این مسأله به بحث می گذاریم.

تخصیص منبع بر اساس هیپ

یک روش این است که به سادگی مشکل پارگی حافظه را نادیده بگیریم و از یک تخصیص دهنده ی هیپ عمومی استفاده کنیم برای تخصیص منابع تان (مانند پیاده‌سازی که توسط malloc() در C یا عملگر جهانی new در C++ انجام می دهد). این به بهترین حالت کار می‌کند اگر بازی شما تنها با هدف اجرا شدن بر روی کامپیوتر های شخصی باشد و همچنین سیستم عامل هایی که از تخصیص حافظه ی مجازی پشتیبانی می کنند. در چنان سیستمی حافظه ی فیزیکی پاره پاره می‌شود اما قابلیت سیستم عامل برای نگاشت بخش‌های غیر همجنس رم فیزیکی به یک حافظه ی هم جنس مجازی کمک می‌کند که بعضی از مشکلات پارگی را کم کرد.

اگر بازی شما بر روی کنسول با رم فیزیکی محدود و تنها یک مدیر حافظه ی اولیه (یا بدون هیچ مدیر) اجرا می شود، در این صورت پارگی یک مشکل ایجاد می کند. در این مورد یک راه حل این است که به صورت دوره ای حافظه را دفرگمنت کرد. ما می‌بینیم که چگونه این کار انجام می‌شود در بخش 5.2.2.2

تخصیص منبع بر اساس پشته

یک تخصیص دهنده ی پشته از مشکلات پارگی رنج نمی‌برد زیرا حافظه به صورت پیوسته تخصیص می‌شود و به ترتیب برعکس آزاد می شود. یک تخصیص دهنده ی پشته می‌تواند برای بارگزاری منابع ستفاده شود اگر دو شرط پایین وجود داشته باشند:

* بازی خطی باشد و تمرکزش بر روی مرحله قرار داشته باشد (مثلاً بازیکن باید یک صفحه ی بارگزاری ببیند سپس یک مرحله را بازی کند سپس یک صفحه ی بارگزاری دیگر را ببیند سپس یک مرحله ی دیگر را ببیند)
* هر مرحله به صورت کامل درون حافظه جای گیرد

با فرض کردن اینکه این نیازمندی ها تأمین شده است ما می‌توانیم از تخصیص دهنده ی پشته برای بارگزاری منابع به این صورت استفاده کنیم: هنگامی که بازی برای بار اول شروع می شود، منابع جهانی در ابتدا تخصیص داده می شوند. بالای پشته سپس نشانه گذاری می‌شود تا اینکه بتوانیم تا این قسمت را بعداً آزاد کنیم. برای بارگزاری یک مرحله ما صرفاً این منبع را بر روی بالای پشته تخصیص می دهیم. هنگامی که مرحله کامل شد ما می‌توانیم صرفاً بالای پشته را تا نشانه ای که پیشتر گذاشتیم آزاد کنیم بنابراین تمام منابع مرحله به یک آن آزاد می‌شوند بدون اینکه منابع جهانی تحت تأثیر قرار بگیرند. این فرایند می‌تواند برای هر تعداد از مراحل تکرار شود بدون پاره کردن حافظه. شکل ۶.۳ چگونگی این کار را نشان می دهد.

یک تخصیص دهنده ی پشته ی دو‌طرفه می‌تواند برای بهتر کردن این روش مورد استفاده قرار گیرد. دو پشته درون یک بلوک بزرگ از حافظه تخصیص داده می شوند. یکی از پایین منطقه ی حافظه رشد می‌کند در حالی که دیگری از بالا به پایین رشد می کند. تا زمانی که دو پشته با همدیگر برخورد نکرده اند پشته می‌تواند به صورت طبیعی منابع حافظه را بگیرد و بازپس دهد-- گاهی اوقات این ممکن نیست اگر هر پشته تنها در همان اندازه ثابت بلوک خود بماند.

در Hydro Thunder، میدوی از یک تخصیص دهنده ی دو‌طرفه استفاده کرد. استک پایینی برای بارگزاری داده‌های پایدار بود در حالی که بالایی برای تخصیص داده‌های موقت بود که در هر فریم آزاد می شدند. یک راه دیگر که یک پشته ی دوطرفه می‌تواند مورد استفاده قرار گیرد، بارگزاری پینگ پونگی مراحل است. چنان روشی در Bionic games مورد استفاده قرار گرفته است. برای یکی از پروژه هایشان. ایده ی اولیه این است که یک نسخه ی فشرده از مرحله ی B را درون پشته ی بالایی فشرده‌سازی کنیم در حالی که مرحله ی فعال کنونی A در پشته ی پایینی است (به صورت غیر فشرده). برای سویچ از مرحله ی A به مرحله ی B، ما صرفاً منابع مرحله ی A را آزاد می‌کنیم (با پاک کردن پشته ی پایینی) و سپس مرحله ی B را از پشته ی بالایی به پشته ی پایینی غیرفشرده سازی می کنیم. غیرفشرده سازی معمولاً بسیار سریعتر از بارگزاری است از دیسک بنابراین این روش به صورت موثری زمان بارگزاری را که ممکن است در غیر این صورت توسط بازیکن در بین مراحل قابل لمس باشد حذف می کند.

تخصیص منابع بر اساس مجموعه

یک تکنیک تخصیص منابع که در موتورهای بازی که استریم کردن را پشتیبانی می کنند، معمول است این است که داده‌های منابع را در بخش‌های مساوی بارگزاری کرد. زیرا بخش‌ها همه به یک اندازه هستند می‌توانند با استفاده از یک تخصیص دهنده ی نظرسنجی تخصیص داده شوند (بخش 5٫2٫1٫2 را ببینید). هنگامی که منابع بعداً خارج می شوند، بخش‌ها می‌توانند بدون ایجاد پارگی آزاد شوند.

البته یک تکنیک تخصیص براساس مجموعه نیازمند این است که تمام داده‌های منابع به گونه‌ای پهن شوند که اجازه ی تقسیم به بخش‌های مساوی را بدهد. ما نمی‌توانیم صرفاً یک منبع دلخواه را در یک بخش بارگزاری کنیم زیرا فایل ممکن است شامل یک ساختار داده ی پیوسته باشد مانند یک آرایه یا یک استراکت بسیار بزرگ که بزرگ‌تر از یک بخش تکی است. برای نمونه اگر بخشی که شامل یک آرایه است به صورت دنباله ای در رم مرتب نشوند، پیوستگی آرایه از دست می‌رود و اندیس گزاری آرایه به درستی عمل نمی کند. این بدین معناست که تمام داده‌های منابع باید با در نظر گرفتن بخشی بودن طراحی شوند. ساختار داده‌های طولانی و پیوسته باید به نفع ساختار داده‌هایی که به اندازه کافی کوچک هستند که درون یک بخش جای گیرند و یا به رم پیوسته برای به درستی عمل کردن نیاز ندارند (مثلاً لیست دارای لینک) کنار گذاشته شوند.

هر بخش در مجموعه مخصوصاً با یک مرحله ی خاص از بازی مرتیط می شود. (یک راه ساده برای انجام این کار این است که به هر مرحله یک لیست لینک شده از بخش هایش بدهیم) این اجازه می‌دهد که موتور زمان حیات هر بخش را به صورت درستی مدیریت کند حتی با وجود اینکه چندین مرحله با چندین طول حیات متفاوت به صورت همزمان در حافظه باشند. برای مثال هنگامی که مرحله ی X بارگزاری می‌شود ممکن است N بخش را تخصیص دهد و از آن‌ها استفاده کند. بعداً مرحله ی Y تعداد M بخش اضافی را تخصیص می دهد. هنگامی که مرحله ی X در نهایت خارج شد، N بخش آن به مجموعه ی آزاد برگردانده می شود. اگر مرحله ی Y هنوز فعال باشد، M بخش آن باید در حافظه باقی بمانند. با ارتباط دادن هر بخش با یک مرحله ی خاص زمان حیات بخش‌ها می‌تواند به سادگی و با کارایی مدیریت شود. این در شکل ۶.۴ نشان داده شده است.

یک مصالحه ی بزرگ در یک شمای تخصیص منابع فضای از دست رفته است. در صورتی که اندازه ی یک فایل منبع یک ضریب دقیق از اندازه ی بخش نباشد، آخرین بخش در یک فایل کاملاً مورد استفاده قرار نخواهد گرفت (بخش ۶.۵ را ببینید). انتخاب یک اندازه بخش کوچکتر می‌تواند به حداقل رساندن این مشکل را یاری دهد اما هر چه بخش کوچکتر باشد، محدودیت‌های بر روی ظاهر داده ی منبع بیشتر می شود. (به عنوان یک مثال افراطی، اگر اندازه بخش با یک بایت انتخاب شود، در این صورت هیچ ساختار داده‌ای نمی‌تواند بزرگ‌تر از یک تک باید باشد – مشخصاً یک وضعیت غیر عملی) یک اندازه بخش معمولی یک ضریب از چند کیبی بایت است. برای نمونه در ناتی داگ ما از یک تخصیص دهنده ی منبع بخشی به عنوان قسمتی از سیستم استریم منبعمان استفاده می‌کنیم و بخش‌های ما ۵۱۲ کیبی بایت بر روی پلی استیشن ۳ و یک میبی بایت بر روی پلی استیشن ۴ هستند. شما ممکن است همچنین بخواهید که یک اندازه بخشی را در نظر بگیرید که ضریبی از اندازه بافر سیستم عامل است تا کارایی را هنگام بارگزاری هر تک بخش بهبود دهید.

تخصیص دهنده ی بخش منبع

یک راه برای محدود کردن تأثیرات حافظه ی از دست رفته ی بخش این است که یک تخصیص دهنده ی خاص حافظه را بسازید که می‌تواند از بخش‌های استفاده نشده از بخش‌ها استفاده کند. تا جایی که من اطلاع دارم، هیچ نام استانداردی برای این نوع از تخصیص وجود ندارد اما میتوان آن را یک تخصیص دهنده ی بخش منبع خواند تا زمانی ک یک نام بهتر پیدا نشده است.

یک تخصیص دهنده ی بخش منبع مخصوصاً پیاده‌سازی دشواری ندارد. ما تنها نیاز به این داریم که یک لینکد لیست از تمام بخش‌هایی که از حافظه مورد استفاده قرار نگرفته آند داشته باشیم، همراه با آن مکان ها و اندازه های هر بلوک خالی را داشته باشیم. ما می‌توانیم سپس از این بلوک های خالی به هر صورتی ک می‌خواهیم استفاده کنیم. برای مثال ما ممکن است لینکد لیست از بلوک های خالی را با استفاده از یک تخصیص دهنده ی هیپ عمومی مدیریت کنیم. یا ما ممکن است یک تخصیص دهنده ی کوچک پشته را بر روی هر بلوک خالی قرار دهیم، هر زمان که یک درخواست برای حافظه می آيد ما می‌توانیم در آن صورت بلوک های خالی را اسکن کرده تا آن‌هایی را پیدا کنیم که پشته شان رم خالی کافی دارد و سپس از آن پشته برای تأمین نیاز استفاده کنیم.

متأسفانه تقریباً یک مگس عجیب در پماد ما در اینجا وجود دارد. اگر ما حافظه را در بخش‌های استفاده نشده از بخش‌های منبعمان تخصیص دهیم، در این صورت چه اتفاقی می‌افتد اگر تمام این بخش‌ها آزاد شوند؟ ما نمی‌توانیم هیچ قسمت از یک بخش را آزاد کنیم – یک وضعیت همه یا هیچ است. بنابراین هر حافظه ای که ما درون یک بخش استفاده نشده از یک بخش منبع تخصیص می‌دهیم به صورت جادویی محو می‌شود در هنگامی که آن منبع خارج می شود.

یک راه حل ساده برای است مشکل این است که تنها از تخصیص دهنده ی بخش- آزاد برای درخواست ها ی حافظه که زمان حیاتشان با زمان حیات مرحله‌ای که یک بخش خاص با آن مرتبط است، استفاده کنیم. به بیان دیگر ما باید تنها حافظه را از مرحله A تخصیص دهیم برای داده‌هایی که به صورت اختصاصی با مرحله ی A مرتبط هستند و تنها حافظه ی بخشی را از بخش‌های B تخصیص دهیم برای مواردی که تنها با مرحله ی ‌B مربوط هستند. این تخصیص دهنده ی بخش منبع ما را مجبور می‌کند که بخش‌های هر منبع را به صورت جداگانه مدیریت کند. و کاربر تخصیص دهنده ی بخش را مجبور می‌کند که معلوم کند کدام مرحله به آن‌ها تخصیص پیدا می‌کند تا اینکه لینکد لیست درست از بلوک های خالی برای ارضای این درخواست مورد استفاده قرار گیرد.

خوشبختانه بیشتر موتور های بازی باید هنگام بارگزاری منابع،‌حافظه را به صورت دینامیک تخصیص دهند، بالا و بر روی حافظه ی مورد نیاز برای خود فایل‌های منابع. تا اینکه یک تخصیص دهنده ی بخش منبع بتواند یک روش مؤثر برای بازگرداندن حافظه ی بخشی باشد که در غیر این صورت ممکن است از دست برود.

فایل‌های منبع قسمتی شده

یک ایده ی کارای دیگر که مرتبط با فایل‌های منبع بخشی است مفهوم تکه‌ای از فایل است. یک فایل منبع معمولی ممکن است شامل بین یک تا چهار تکه باشد هر یک از آن‌ها بین یک یا بیشتر بخش برای هدف تخصیص مجموعه همان‌طور که در بالا توضیح داده شد تقسیم شده است. یک بخش ممکن است شامل داده‌هایی باشد که با هدف رم اصلی است در حالی که دیگری ممکن است شامل داده‌های رم ویدیویی باشد. یک تکه ی دیگر می‌تواند شامل داده‌های موقت باشد که در طول فرایند بارگزاری مورد نیاز است اما آن‌هایی را که کاملاً بارگزاری شده است رها می کند. هنوز یک بخش دیگر ممکن است شامل اطلاعات دیباگ کردن باشد. این داده‌های دیباگ می‌تواند هنگام اجرای بازی در وضعیت دیباگ بارگزاری شود اما در تولید نهایی بازی به هیچ عنوان بارگزاری نشود. سیستم فایل SDK متعلق به Granny یک مثال عالی از این است که چگونه اسکن فایل را به یک صورت ساده و قابل انعطاف پیاده‌سازی کرد.

6.2.2.8 منابع مرکب و تمامیت ارجاعی

معمولاً یک پایگاه داده ی منابع بازی شامل چندین فایل منبع است که هر کدام شامل یک یا بیشتر اشیاء داده‌ای هستند. این اشیای داده‌ای می‌توانند به همدیگر ارجاع شوند و به هم وابسته باشند در هر شیوه ی دلخواه. برای نمونه یک ساختار داده ی مش ممکن است شامل یک ارجاع به موادش باشد که خود شامل یک لیست از ارجاعات به بافت است. معمولاً ارجاع درونی به معنی وابستگی است (یعنی اگر منبع A به منبع B ارجاع دهد، در این صورت هر دوی A و B باید در حافظه باشند تا منابع در بازی بتوانند کار کنند). در کل یک پایگاه داده ی منابع بازی می‌تواند با یک گراف جهت دار از اشیای داده‌ای وابسته نمایش داده شود.

ارجاع درونی بین اشیای داده‌ای م یتواند درونی باشد (یک ارجاع بین دو شی درون یک فایل) و یا بیرونی (یک ارجاع به یک شی در یک فایل دیگر). این تفاوت مهم است زیرا ارجاعات درونی و بیرونی معمولاً به صورت متفاوتی پیاده‌سازی می شوند. هنگام بصری سازی یک پایگاه داده ی بازی ما می‌توانیم خطوط نقطه نقطه را در اطراف فایل‌های منابع تکی رسم کنیم تا تفاوت درونی/بیرونی را مشخص کنیم – هر یال از گراف که از یک خط نقطه نقطه عبور کند یک ارجاع خروجی است در حالی که یال هایی که از این محدوده ها عبور نمی‌کنند درونی هستند. این در شکل 6.6 نمایش داده شده است.

ما گاهی اوقات از عبارت منبع مرکب استفاده می‌کنیم تا یک خوشه خود متکامل از منابع با وابستگی‌های میانی را توضیح دهیم. برای نمونه یک مدل، یک منبع مرکب از یک یا بیشتر مش های مثلثی، یک اسکلت اختیاری و یک