Vol. 6, No. 18, Apr-Jun 2026 (17-40)
Journal of New Achievements in Electrical, Computer and Technology (JNAECT)
مجله دستاوردهای نوین در برق،کامپیوتر و فناوری
دوره ۶، شماره ۱۸، بهار ۱۴۰۵ (۴۰-۱۷)
journal homepage: https://jnaect.ir

تسریع فرآیند ساخت بازی: استفاده از پکیج های کدنویسی آماده در انجین یونیتی

امیرحسین اردلان*
استاد بازیسازی کالج بین المللی ابن سینا، تهران، ایران. تفلیس، گرجستان(shsbbhorse@chmail.ir) .

چکیده

ساخت بازی‌های رایانه‌ای فرآیندی پیچیده و چندمرحله‌ای است که نیازمند همکاری تیم‌های تخصصی در زمینه‌های مختلف می‌باشد. برنامه‌نویسی، به‌عنوان یکی از مراحل اساسی و زمان‌بر، نقش مهمی در سرعت و کیفیت تولید بازی ایفا می‌کند. استفاده از پکیج‌های کدنویسی آماده در موتور بازی‌سازی یونیتی، امکان کاهش قابل‌توجه زمان برنامه‌نویسی را فراهم می‌آورد و بازیسازان را قادر می‌سازد تا به جای نوشتن کدهای تکراری، بر روی نوآوری و بهبود تجربه کاربری تمرکز کنند. این مقاله به بررسی نقش و مزایای این پکیج‌ها پرداخته و نمونه‌هایی از کاربرد آنها را در پروژه‌های مختلف تحلیل می‌کند. بهره‌گیری از این روش، نه‌تنها در استودیوهای بزرگ، بلکه در تیم‌های کوچک نیز متداول است و امکان مدیریت بهتر زمان، تمرکز بیشتر بر جنبه‌های داستانی و هنری بازی و همچنین افزایش بهره‌وری را فراهم می‌سازد. این پژوهش بر اساس تجربیات عملی و با استناد به آموزش‌های اساتید برجسته در زمینه پکیج‌نویسی در انجین یونیتی (Unity Engine) تدوین شده و به بررسی دقیق بخش‌های مختلف این پکیج‌ها، از جمله قابلیت‌های فنی و مزایای اقتصادی آنها می‌پردازد.

واژه‌های کلیدی: بازیسازی، یونیتی، Unity، برنامه نویسی، بازی رایانه ای، بهره‌وری، پکیج سازی، هوش مصنوعی، AI

۱. مقدمه

صنعت بازی‌سازی، امروزه با رشد چشمگیری مواجه است. نیاز به تولید سریع‌تر و باکیفیت‌تر بازی‌ها بیش از پیش احساس می‌شود. یکی از چالش‌های اصلی در این مسیر، زمان‌بر بودن فرآیند برنامه‌نویسی و نیاز به دانش فنی عمیق است که می‌تواند مانعی برای بازیسازان مبتدی یا تیم‌های کوچک باشد[۱]. در این میان، استفاده از پکیج‌های کدنویسی آماده در موتور بازی‌سازی یونیتی به‌عنوان راهکاری کارآمد مطرح شده است. این پکیج‌ها، با ارائه ابزارها و کدهای از پیش نوشته شده؛ امکان پیاده‌سازی سریع ویژگی‌های مختلف بازی را فراهم می‌آورند و بازیسازان را از نوشتن کدهای تکراری و پیچیده بی‌نیاز می‌سازند.

یکی از مزایای اصلی این پکیج‌ها، ساده‌سازی فرآیند توسعه است. به‌طوری‌که حتی افراد مبتدی نیز می‌توانند با تنظیم پارامترهای ساده یک پکیج در پنجره مورد نظر یعنی Inspector، بدون نیاز به دانش برنامه‌نویسی پیشرفته، به نتایج حرفه‌ای دست یابند. این امر نه‌تنها زمان تولید را به‌طور قابل‌توجهی کاهش می‌دهد بلکه هزینه‌های توسعه را نیز بهینه می‌سازد و امکان تمرکز بیشتر بر جنبه‌های خلاقانه و هنری بازی را فراهم می‌آورد. بسته به نوع و پیچیدگی بازی، استفاده از این پکیج‌ها می‌تواند از چند روز تا چندین هفته در مدت زمان برنامه‌نویسی صرفه‌جویی کند. علاوه بر این، استفاده از پکیج‌های آماده و ساخت آنها، خود به‌عنوان یک شغل تخصصی در صنعت بازی‌سازی مطرح شده است و فرصت‌های جدیدی را برای برنامه‌نویسانی ایجاد کرده است که به طراحی و فروش این پکیج‌ها علاقه‌مند هستند.

در این مقاله، به بررسی دقیق نقش و مزایای پکیج‌های کدنویسی آماده در یونیتی پرداخته می‌شود و نمونه‌هایی از کاربرد آنها در پروژه‌های مختلف تحلیل می‌گردد. هدف اصلی، نشان دادن تأثیر مثبت این ابزارها بر سرعت، کیفیت و بهره‌وری در فرآیند ساخت بازی است.

۲. روش تحقیق

این پژوهش با استفاده از روش «تحقیق کتابخانه‌ای و تجربی» انجام شده است. در مرحله اول، منابع معتبر خارجی شامل مقالات، مستندات رسمی یونیتی، آموزش‌های اساتید برجسته در زمینه پکیج‌نویسی، و نمونه‌های موفق از پروژه‌های استفاده‌کننده از پکیج‌های آماده در Asset Store) بررسی شده است. در این راستا، به تحلیل ویژگی‌های فنی، مزایای اقتصادی و تأثیر این پکیج‌ها بر سرعت و کیفیت توسعه بازی پرداخته شده است.

در مرحله دوم، با تکیه بر تجربیات عملی و مصاحبه با بازیسازان ایرانی و بین‌المللی، کاربردهای عملی پکیج‌های آماده در پروژه‌های واقعی مورد ارزیابی قرار گرفته است. همچنین، با بررسی نمونه‌های موفق استودیوهای کوچک و بزرگ، تأثیر استفاده از این پکیج‌ها بر کاهش زمان و هزینه‌های توسعه، افزایش بهره‌وری، و بهبود تجربه کاربری تحلیل شده است.

به دلیل عدم وجود منابع داخلی جامع در این زمینه، تمرکز اصلی پژوهش بر منابع خارجی، مستندات رسمی شرکت یونیتی و تجربیات عملی بازیسازان و مؤلف بوده است. بر همین اساس، در جدول ۱ به اصلی‌ترین اهداف ساخت پکیج اشاره شده است.

جدول ۱- اهداف اصلی ساخت پکیج‌های بازیسازی

دلیل ساخت پکیج	توضیح مختصر	اهداف اصلی
انجام پروژه‌های شخصی	برای تسریع و ساده‌سازی فرآیند توسعه بازی‌های شخصی یا آزمایش ایده‌های جدید.	کاهش زمان توسعه، افزایش کیفیت، یادگیری و آزمایش ترفندهای جدید
دورکاری (فریلنسری)	ایجاد پکیج‌های سفارشی برای مشتریان یا پروژه‌های دورکاری به منظور پاسخگویی سریع‌تر به نیازها.	افزایش بهره‌وری، جذب مشتری بیشتر، ایجاد نمونه کار حرفه‌ای
استفاده در استودیو	ساخت پکیج‌های داخلی برای استفاده تیمی و استانداردسازی فرآیندهای توسعه در استودیو.	کاهش هزینه‌ها، افزایش سرعت تولید، یکپارچگی تیمی، مدیریت بهتر پروژه‌ها
فروش در سایت‌هایی مانند Asset Store	تولید پکیج‌های عمومی و فروش آنها به دیگر بازیسازان برای کسب درآمد.	کسب درآمد غیرفعال، شناخته شدن در جامعه توسعه‌دهندگان، دریافت بازخورد و

		بهبود محصول
آموزش و اشتراک‌گذاری دانش	ساخت پکیج‌های آموزشی برای تدریس یا نمونه کد برای کمک به دیگران و اشتراک‌گذاری تجربیات.	کمک به بازیسازان، ایجاد شبکه حرفه‌ای، تدریس، افزایش اعتبار شخص مدرس

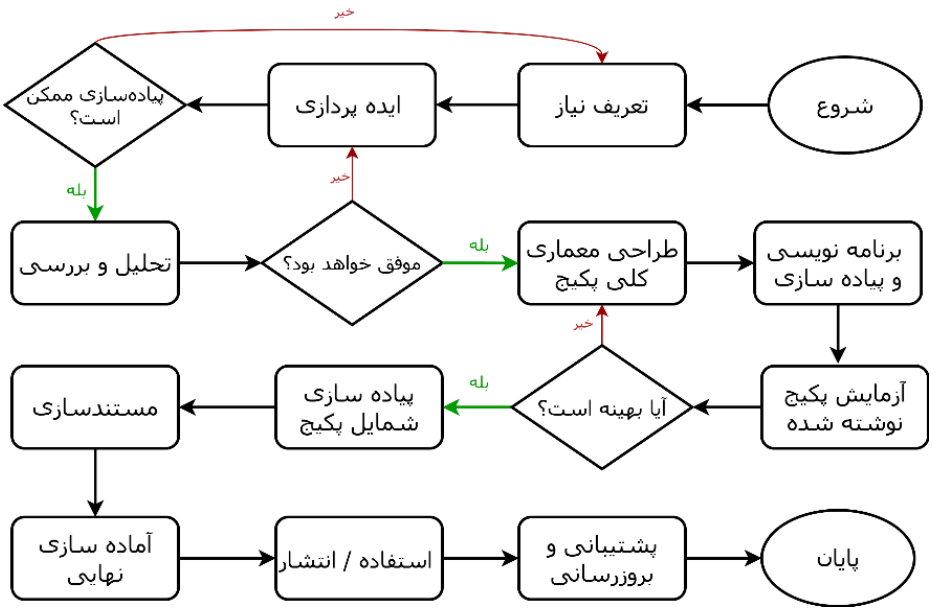
۳. یافته‌ها

یافته‌ها به چند بخش مختلف تقسیم می‌شوند تا تمامی مبانی به صورت کامل درج شوند. در راستای هدف اصلی این مقاله که تسریع فرآیند ساخت بازی با بهره‌گیری از پکیج‌های کدنویسی آماده در انجین یونیتی است؛ بخش یافته‌ها به ارائه تحلیل‌های عملی و نتایج حاصل از بررسی‌های انجام شده می‌پردازد. این بخش، با تبیین دلایل اصلی پیدایش و توسعه پکیج‌های آماده، چارچوبی برای درک اهمیت آنها در کاهش پیچیدگی و افزایش سرعت توسعه فراهم می‌کند. سپس، با معرفی و بررسی چندین پکیج کاربردی، کارکردها و مزایای عملی آنها به صورت موردی تشریح خواهد شد. در ادامه، جنبه‌های مختلف فرآیند طراحی و پیاده‌سازی پکیج کارآمد، از مراحل اولیه تا انتشار، مورد کاوش قرار می‌گیرد تا دیدگاهی جامع از چرخه حیات و کاربرد این ابزارها در اکوسیستم توسعه بازی‌ها ارائه شود.

۱.۳. تاریخچه و روند تکامل پکیج‌های آماده در یونیتی

صنعت بازی‌سازی همواره با چالش‌های زمانی و فنی مواجه بوده است. با گسترش موتورهای بازی‌سازی مانند یونیتی، نیاز به ابزارهایی برای تسریع فرآیند توسعه احساس شد. نرم‌افزار یونیتی در سال ۲۰۰۵ به صورت رسمی منتشر شد[2] و رفته‌رفته، نیاز به پکیج‌ها بیش از پیش برجسته شدند. به همین دلیل از اوایل دهه ۲۰۱۰ پکیج‌نویسی در بازیسازی و یونیتی شروع شد و در مدت زمان کمی فراگیر شد زیرا استودیوها این امر را برای تولید الزام می‌دانستند. با استفاده از پکیج‌ها، در زمان، بودجه و تعداد کارکنان، صرفه‌جویی به عمل آمد و حتی ایرادات بازی‌ها نیز کمتر شد. همه این دلایل باعث شد تا بازی‌ها بهتر بفروشند و در نهایت سودآوری بیشتری داشته باشند. در ابتدا، این پکیج‌ها بیشتر توسط بازیسازان مستقل و برای نیازهای خاص ایجاد می‌شدند. به عنوان مثال صرفا برای نیازهای داخلی استودیوها استفاده میشد اما کم‌کم فراگیر شده و در سایت‌های اینترنتی نیز به فروش رسیدند. با رشد سایت Asset Store که متعلق به خود شرکت یونیتی می‌باشد؛ سازوکاری گسترده برای خرید و فروش این ابزارها شکل گرفت. امروزه، پکیج‌های آماده نه‌تنها توسط افراد، بلکه توسط استودیوهای بزرگ نیز تولید و استفاده می‌شوند و نقش مهمی در کاهش هزینه‌ها و افزایش سرعت تولید بازی ایفا می‌کنند.

طبق نمودار ۱، مراحل ساخت یک پکیج به صورت مرحله به مرحله نمایش داده شده است. نمودار۱ فرآیند طراحی، توسعه، آزمایش و انتشار پکیج را در قالب گام به گام نشان می‌دهد و به درک بهتر چرخه حیات یک پکیج کمک می‌کند.



نمودار ۱. مراحل طراحی، ساخت و انتشار یک پکیج

یکی از دلایل اصلی پیدایش پکیج‌های آماده، نیاز به کاهش کدهای تکراری و ارائه راهکارهای استاندارد برای مشکلات رایج در توسعه بازی است. به‌عنوان مثال، پکیج‌هایی مانند ODIN Inspector و Dotween، با ارائه ابزارهای پیشرفته برای مدیریت داده‌ها و انیمیشن‌ها، امکان پیاده‌سازی سریع ویژگی‌های پیچیده را فراهم کرده‌اند. این پکیج‌ها، با ارائه کدها و ابزارهای از پیش نوشته شده؛ بازیسازان را قادر می‌سازند تا بدون نیاز به دانش برنامه‌نویسی پیشرفته، به نتایج حرفه‌ای دست یابند.

نکات کلیدی برای موفقیت در ساخت پکیج:

نیازسنجی دقیق: قبل از شروع، نیازهای واقعی بازار را شناسایی کنید.

کیفیت و بهینه‌سازی: پکیج باید عملکردی بهینه و کیفیت بالایی داشته باشد.

مستندسازی کامل: مستندسازی خوب، استفاده از پکیج را برای کاربران آسان‌تر می‌کند.

پشتیبانی فعال: پشتیبانی و بروزرسانی منظم، اعتماد کاربران را جلب می‌کند.

بازاریابی موثر: برای موفقیت پکیج، باید استراتژی بازاریابی مناسبی داشت.

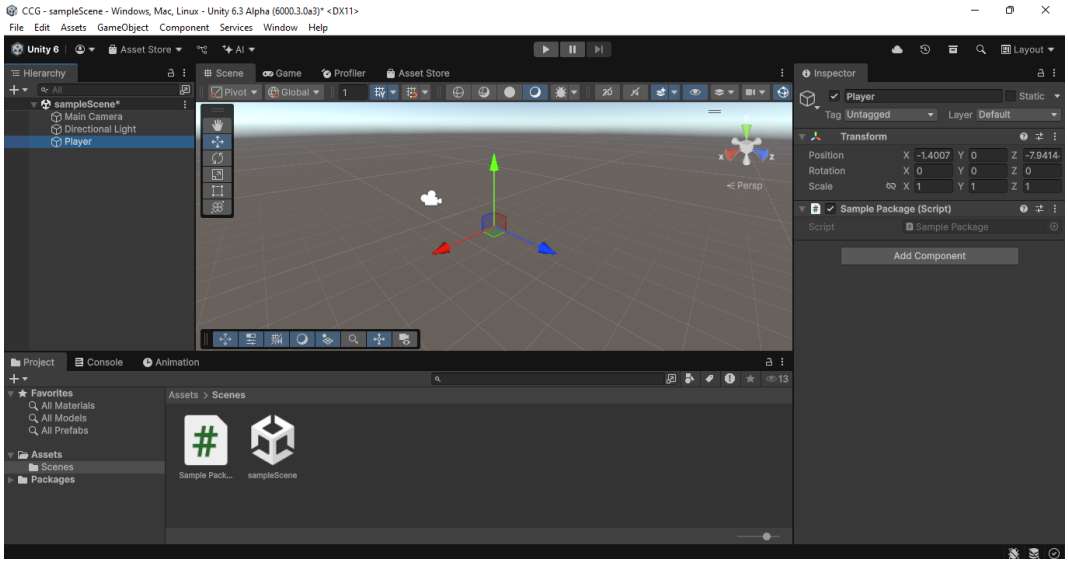
بسیاری از گروه‌های بازیسازی ترجیح می‌دهند صرفا پکیج بسازند. زیرا توانایی خود را بدین صورت، بهتر نمایش می‌دهند. در برخی کشورها نیز که استخدام بازیسازان با مشکل مواجه می‌شود؛ ساخت پکیج و معرفی بین‌المللی می‌تواند گزینه بسیار مناسبی باشد.

۲.۳. معماری و اجزای اصلی پکیج‌های کدنویسی آماده

یکی از مهم‌ترین جنبه‌های ساخت پکیج‌های کدنویسی آماده در یونیتی، طراحی رابط کاربری مناسب در پنجره Inspector است[3]. این پنجره به عنوان اصلی‌ترین نقطه تعامل بازیسازان با پکیج، نقش کلیدی در تجربه کاربری و سهولت استفاده را ایفا می‌کند. یک رابط کاربری خوب نه تنها باعث می‌شود پکیج حرفه‌ای‌تر به نظر برسد بلکه کارایی و سرعت توسعه را نیز افزایش می‌دهد.

در این بخش، به بررسی معماری و اجزای اصلی پکیج‌های کدنویسی آماده پرداخته می‌شود و روش‌های ایجاد زیبایی‌ها و بخش‌های مختلف در Inspector به تفصیل شرح داده می‌شود. این شامل استفاده از ابزارها و تکنیک‌های مختلف برای سازماندهی، نمایش، و بهبود تعامل با پارامترها و تنظیمات پکیج است.

همانطور که در شکل ۱ مشاهده می‌شود؛ محیط داخلی نرم‌افزار یونیتی نمایش داده شده است. پنجره Inspector که در سمت راست تصویر قرار دارد؛ یکی از بخش‌های اصلی یونیتی برای ویرایش و تنظیم پارامترهای اشیاء بازی است. با تغییر کد نوشته شده؛ بخش‌های مختلف برای مدیریت پکیج به همین پنجره و در قسمت کد مورد نظر اضافه خواهند شد. مهم‌ترین ابزارهای نمایشی بررسی شده‌اند اما ابزارهای بیشتری نیز وجود دارند اما مستلزم به استفاده از ODIN Inspector و یا دیگر ابزارها هستند. در مقاله حاضر، صرفا ابزارهایی بررسی شده‌اند که بدون نیاز به ابزار خاصی و صرفا با اصل یونیتی قابل اجرا هستند. درست است که زیبایی پکیج اهمیت بسیار بالایی دارد اما نباید سنگین باشد وگرنه امتیاز بهینه بودن را از دست خواهد داد.



شکل ۱. محیط داخلی نرم‌افزار یونیتی

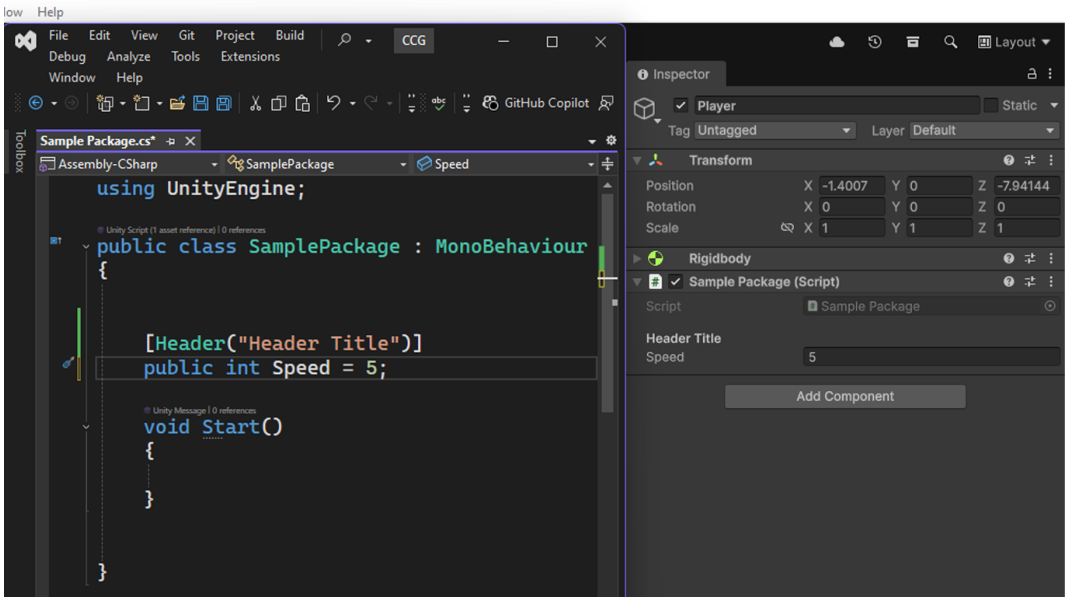
- سربرگ Header

سربرگ یکی از ابزارهای پرکاربرد است. تعیین محدوده فعالیت و هدف اصلی هر بخش یا ابزاررا بیان می‌کند. یک پکیج می‌تواند دارای چندین سربرگ باشد و محدودیتی از این لحاظ ندارد. در رابطه ۱ کد مرتبط با سربرگ نشان داده شده است.

نکته۱: اصل ترتیب نمایش بر اساس استاندارد برنامه‌نویسی سی-شارپ (C#) می‌باشد. یعنی خط به خط و از خطوط بالا به سمت پایین است. بر همین اساس در پنجره inspector نیز نمایش داده خواهند شد.

نکته۲: سربرگ حتما باید دارای زیرمجموعه باشد وگرنه در شرایطی که زیرمجموعه یا عضوی نداشته باشد؛ خطا خواهد داد و نمایش داده نخواهد شد. به همین دلیل در شکل۲ یک متغیر نیز قرار داده شده است. در بخش سمت راست نحوه نمایش در یونیتی و در سمت چپ کدهای نوشته‌شده نمایش داده شده‌اند.

```
[Header ("نام مورد نظر سربرگ")]                    (1)
```

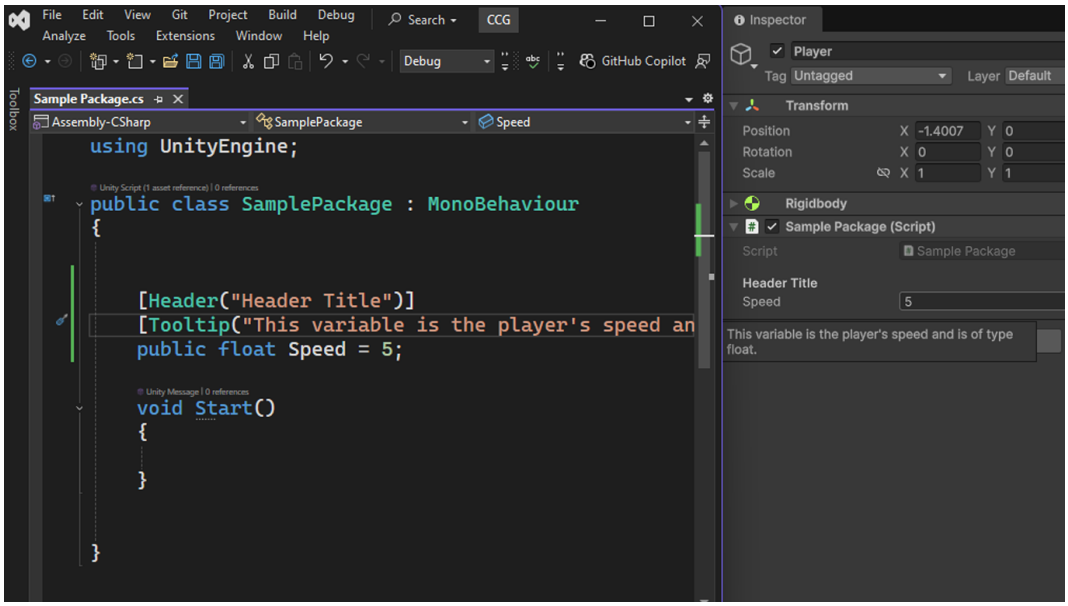


شکل ۲. نمونه‌ای از استفاده از سربرگ در یونیتی

- راهنمای ابزار (ToolTip)

راهنمای ابزار به عنوان توضیحی مختصر برای متغیرها یا بخش‌های مختلف قرار می‌گیرد. زمانی که موس روی متغیر مورد نظر باشد؛ بعد از کمی مکث، نوشته‌ای نمایش داده خواهد شد. این نوشته یک راهنما برای استفاده از ابزار یا متغیر مورد نظر است. به عنوان مثال یک راهنمایی در شکل ۳ نمایش داده شده و استفاده از این ابزار نیز به صورت ساده در رابطه ۲ نمایش داده شده است.

```
[Tooltip ("توضیحات راهنمای ابزار")]                    (2)
```



شکل ۳.استفاده از راهنمای ابزار برای متغیر

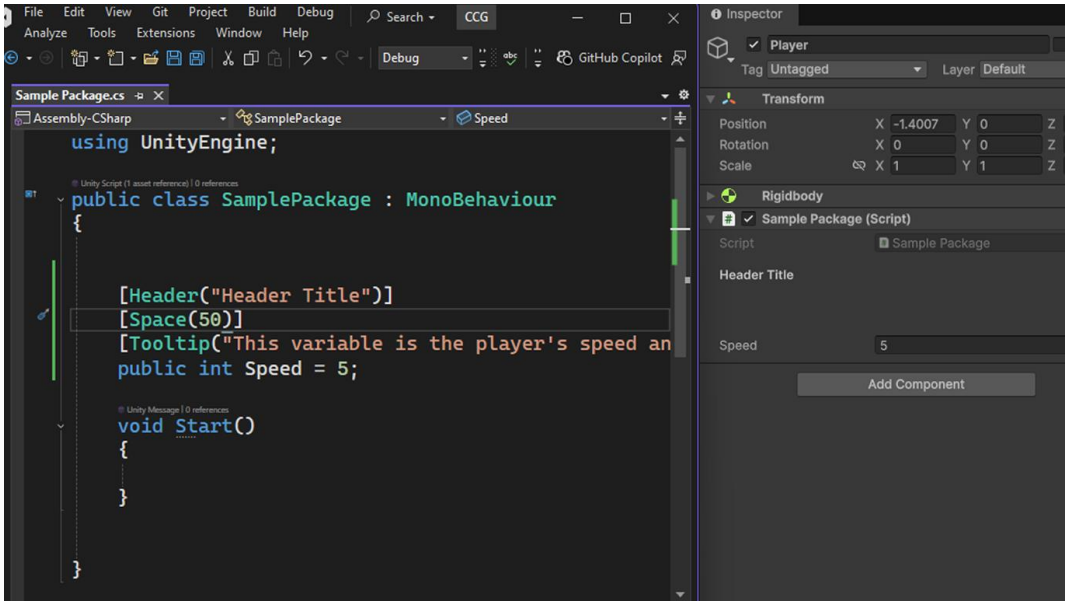
نکته: راهنمای ابزار بر روی متغیری اعمال خواهد شد که بعد از خود راهنمای ابزار نوشته شده باشد. اگر چندین ابزار هم پشت سرهم تعریف شده باشند؛ هیچ مشکلی پدیدار نخواهد شد. چندین ابزار نمایشی می‌توانند بر روی یک متغیر پیاده شوند. به این شرط که ابزارها تکراری

نباشند یا ناسازگاری نداشته باشند. مثلا نباید دو راهنمای ابزار برای یک متغیر تعیین گردد زیرا نمی‌توان دو راهنمای ابزار برای یک متغیر داشته باشیم و این امری طبیعی است.

- فاصله (Space)

برای ایجاد فاصله بین بخش‌های مختلف استفاده می‌شود. نمونه آن در رابطه ۳ و شکل ۴ نمایش داده شده است.

```
[Space(عدد مورد نظر براساس پیکسل)]
```
(3)

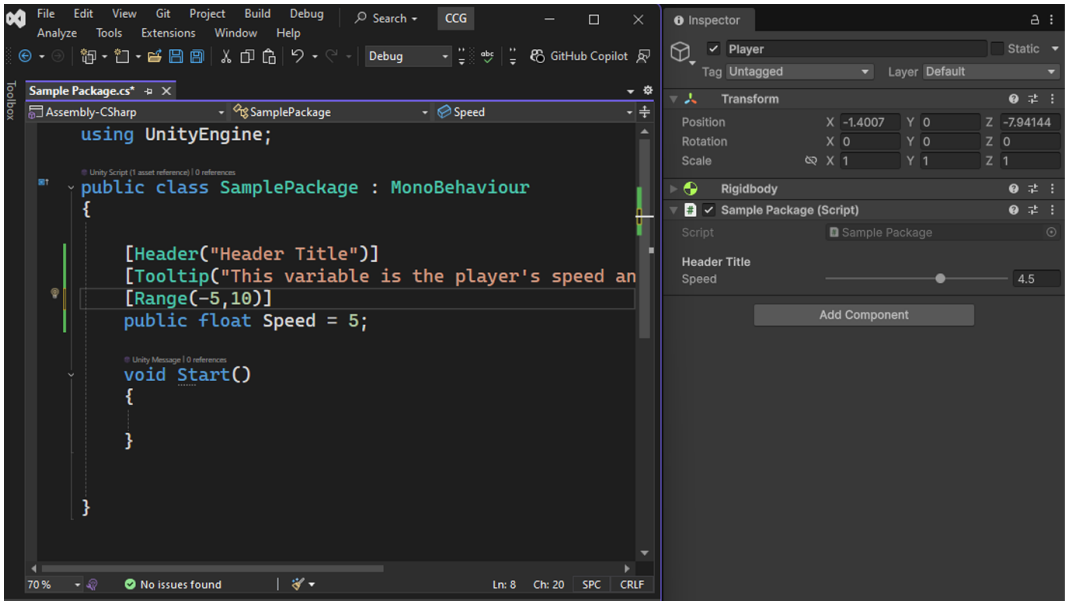


شکل ۴. استفاده از فاصله ۵۰ واحدی بین سربرگ و متغیر

- محدوده (Range)

برای ایجاد یک محدوده حرکتی استفاده می‌شود. برای متغیرهای عددی مانند عددصحیح (int) یا عدداعشاری (float) مورد استفاده قرار می‌گیرد. در رابطه ۴ کد مورد نظر قرار گرفته است. نحوه نمایش آن به صورت یک اسلاید خواهد بود که در شکل ۵ نمایش داده شده است.

```
[Range(حد اقل,حد اکثر)]
```
(4)



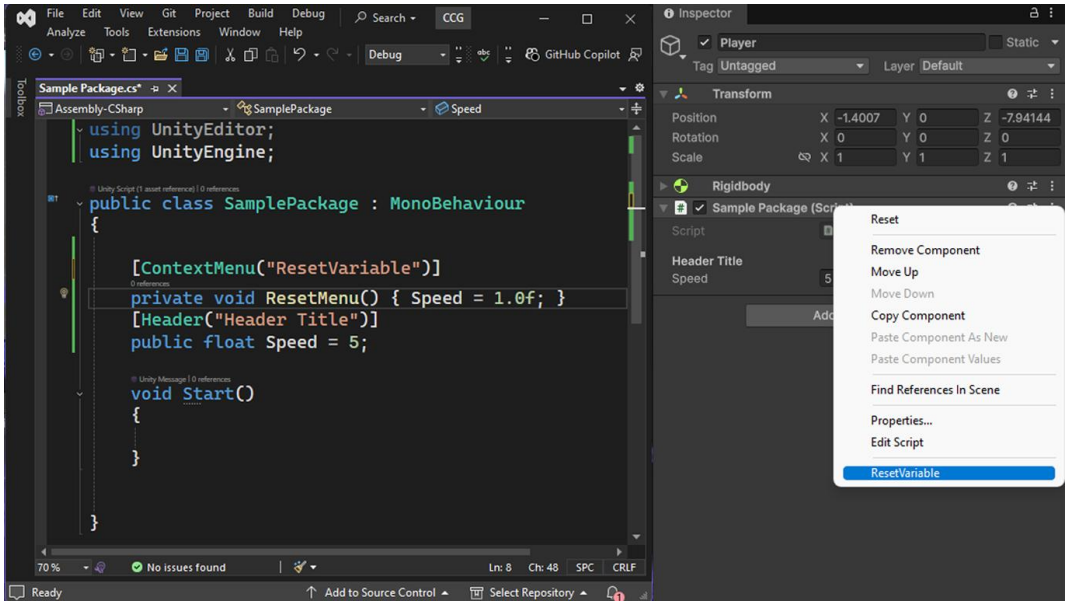
شکل ۵. استفاده از یک محدوده عددی که ۵- تا ۱۰+ تعیین شده است.

نکته: با استفاده از این محدوده، منطقی است که دیگر عدد آنرا از طریق کدنویسی تغییر ندهند. پس باید محدوده اعداد را مناسب و منطقی در نظر گرفت.

- ایجاد گزینه در فهرست (ContextMenu)

با راست-کلیک بر روی کامپوننت مورد نظر، که در اینجا همان کد ما می‌باشد؛ گزینه‌های مختلفی را به ما نمایش می‌دهد. در ساخت پکیج، می‌توان چند گزینه نیز به دلخواه اضافه نمود تا کارهای مختلف را انجام دهند. در رابطه ۵ یک نمونه از این کدها در نظر گرفته شده است.

```
[ContextMenu("اسم مورد نظر")]
void examplename() { کدهای مورد نظر اینجا نوشته شوند }          (5)
```



شکل ۶. در این مثال، گزینه‌ای برای بازگرداندن متغیر به حالت پیشفرض در نظر گرفته شده است.

نکته: گزینه ایجادشده حتما باید یک تابع (void) را اجرا کند.
نکته۲: ممکن است کتابخانه UnityEditor در برخی ابزارها اضافه شوند. اگر به صورت خودکار اضافه نشد و کار با خطا روبرو شد؛ یعنی باید به صورت دستی آنرا اضافه نمود. در شکل ۶ یک نمونه استفاده از ایجاد گزینه نمایش داده شده است.
قبل از معرفی دیگر مقادیر، باید کلیاتی درباره نمایش متغیرها در پنجره inspector ذکر شود. در یونیتی با ۷ پیش‌وند مختلف، امکان تعریف یک متغیر وجود دارد. این پیشوندها هرکدام کارایی و ویژگی‌های مهم خود را دارند که در جدول ۲ برخی ویژگی‌های اصلی‌شان معرفی شده است.

جدول ۲- پیشوندهای متغیر در یونیتی و زبان برنامه نویسی سی‌شارپ

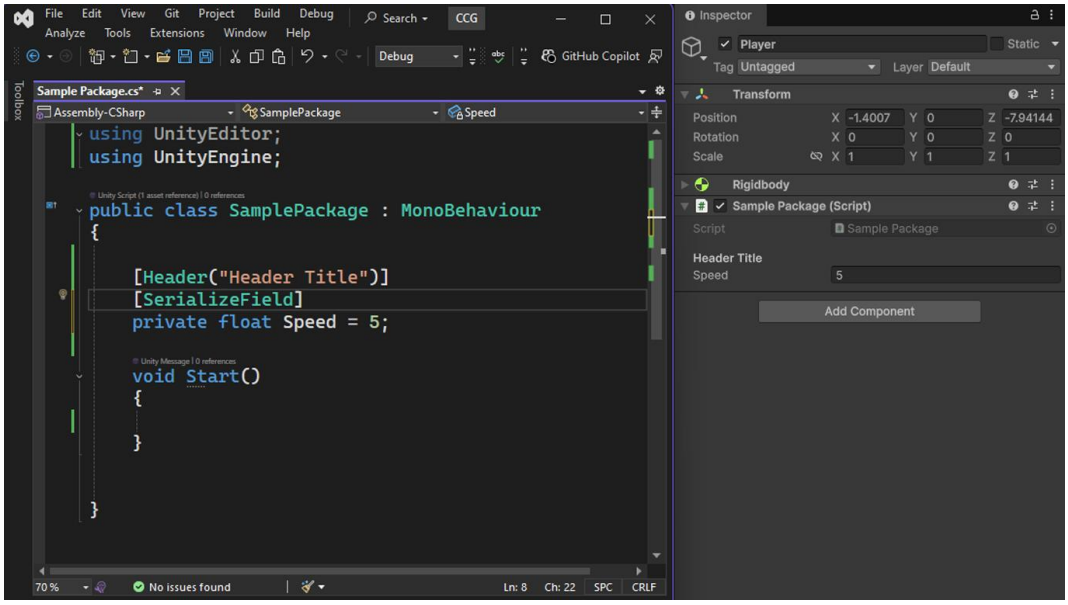
نوع دسترسی	نمایش inspector	توضیح استفاده
عمومی / public	بله	متغیر در همه کلاس‌ها و کدها و دیگر اشیاء به صورت کامل قابل دسترسی است.
خصوصی / private	خیر	متغیر فقط در داخل کلاس خود در دسترس است.
بدون پیشوند	خیر	به صورت پیش‌فرض به حالت خصوصی در خواهد آمد. مگر تنظیمات پروژه متفاوت باشد.
محفوظ / protected	خیر	متغیر فقط در کلاس خود و کلاس‌های مشتق‌شده قابل دسترسی است.
داخلی / internal	خیر	متغیر فقط در داخل Assembly خود قابل دسترسی است.
داخلی محفوظ / protected internal	خیر	متغیر داخل همان اسمبلی و همچنین در کلاس‌های مشتق شده از آن قابل دسترسی است.
خصوصی محفوظ / private protected	خیر	متغیر فقط در داخل کلاس خود و فقط برای کلاس‌های مشتق شده از همان اسمبلی قابل دسترسی است.

طبق جدول ۲ در حالت پیشفرض فقط متغیرهایی که پیشوند عمومی دارند؛ در پنجره inspector نمایش داده خواهند شد. هرکدام از این پیشوندها قابلیت‌هایی در برنامه‌نویسی دارند. برای ساخت پکیج گاها نیاز است که یک متغیر در یونیتی دیده شود اما پیشوند آن خصوصی باشد. یا گاها نیاز است یک متغیر عمومی باشد اما در محیط داخلی یونیتی نمایش داده نشود. به همین جهت ابزارهایی برای این امر وجود دارند.

- SerializeField

استفاده از این ابزار باعث می‌شود تا متغیر مورد نظر در یونیتی نمایش داده شود. مانند دیگر ابزارها، باید رابطه آن یعنی رابطه ۶ را قبل از متغیر مورد نظر نوشت. بدین صورت متغیر مورد نظر حتی اگر خصوصی باشد؛ نمایش داده خواهد شد. این امر در شکل ۷ نمایش داده شده است. هاردمن (۲۰۲۴) در فصل‌های «Menus and UI [7]» و «Build Mode UI [8]» به بررسی ابزارهای بهبود تجربه کاربری در پنجره Inspector نیز می‌پردازد. UI به همان رابط کاربری در محیط بازی گفته می‌شود. مانند دکمه‌ها، منوها، اسلایدرها و دیگر ارکان نمایشی که برای رابط کاربری استفاده می‌شوند.

```
[SerializeField]                                    (6)
```

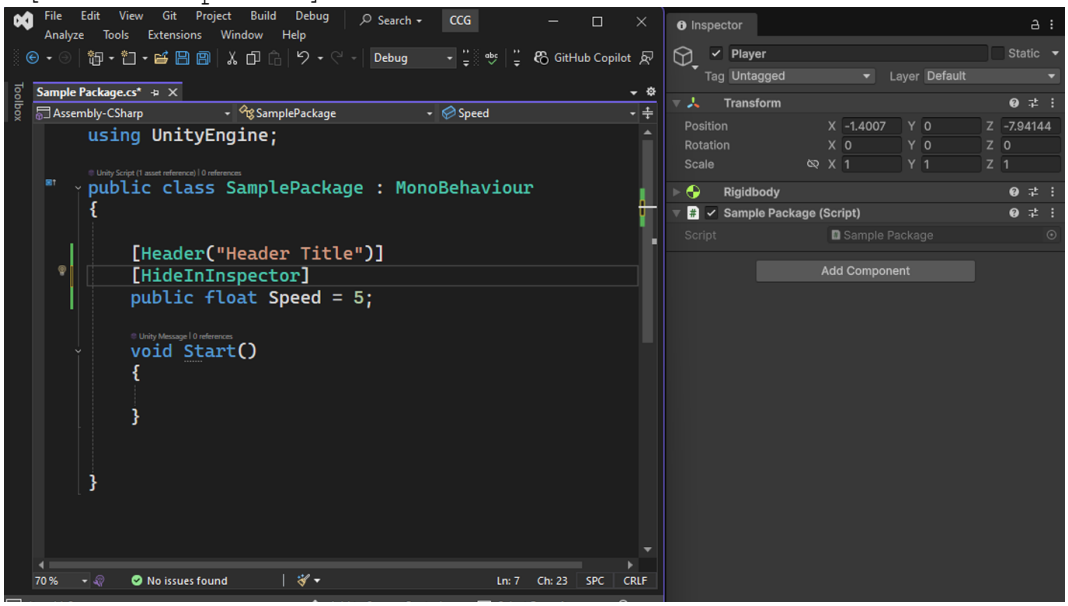


شکل ۷. متغیر مورد نظر با اینکه خصوصی است اما نمایش داده شده است.

- مخفی کردن متغیر (HideInInspector)

استفاده از این ابزار مانند رابطه ۷ باعث می‌شود تا متغیر مورد نظر مخفی بماند حتی اگر از نوع عمومی باشد. استفاده از این ابزار همانند شکل ۸ موجب مخفی ماندن متغیر عمومی می‌شود.

```
[HideInInspector]
```
(7)



شکل ۸. با اینکه متغیر مورد نظر عمومی است اما نمایش داده نشده.

نکته: ممکن است مخفی شدن متغیر، باعث شود تا سربرگ نیز مخفی شود. در شرایطی که سربرگ برای متغیر یا متغیرهایی باشد و آن/آنها مخفی شوند؛ دیگر سربرگ آنها نیز نمایش داده نخواهد شد.

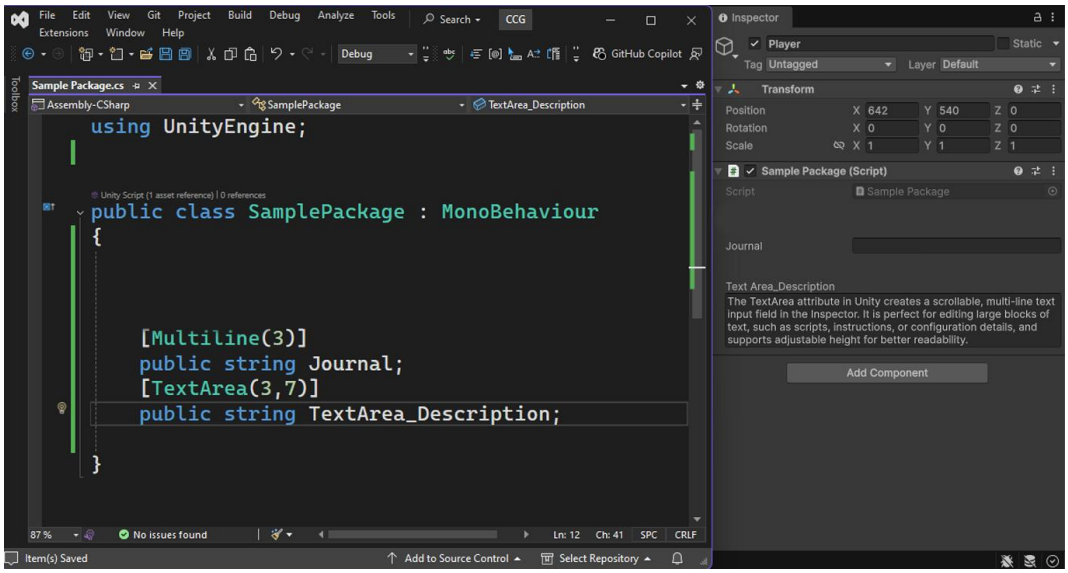
- Multiline

برای ساخت نوشته‌های کمی طولانی‌تر و چندخطی استفاده می‌شود. طبق رابطه ۸ می‌توان از آن استفاده نمود.

- محدوده نوشته TextArea

با این ابزار می‌توان محدوده‌های مختلفی برای نوشته‌ها ایجاد نمود. مخصوصا برای نوشته‌های طولانی که نیاز است محدوده آن مهار شود. مثلا توضیحات طولانی یا نوشته‌های مختلف در بازی‌ها مخصوصا بازی‌های آنلاین را راحت‌تر می‌توان مهار کرد. طبق رابطه ۹ می‌توان از آن استفاده نمود و تعداد خطوط آن را نیز تعیین کرد. در شکل ۹ هر دو ابزار مرتبط با نوشته‌ها نمایش داده شده است.

```
[Multiline(اندازه محدوده نوشته)]                    (8)
[TextArea(حداکثر تعداد خطوط,تعداد خطوط)]            (9)
```



شکل ۹. نمایش متغیرهایی رشته‌ای (حروفی) با استفاده ابزارهای نمایش

نکته: حداکثر تعداد خطوط می‌تواند بسیار کمک‌کننده باشد. اگر تعداد خطوط از حداکثر تعیین شده بالاتر برود؛ خود محیط نرم‌افزار برای آن یک محدوده همراه با اسکرول قرار می‌دهد تا بیشتر از میزان حداکثر جایی را اشغال نکند.

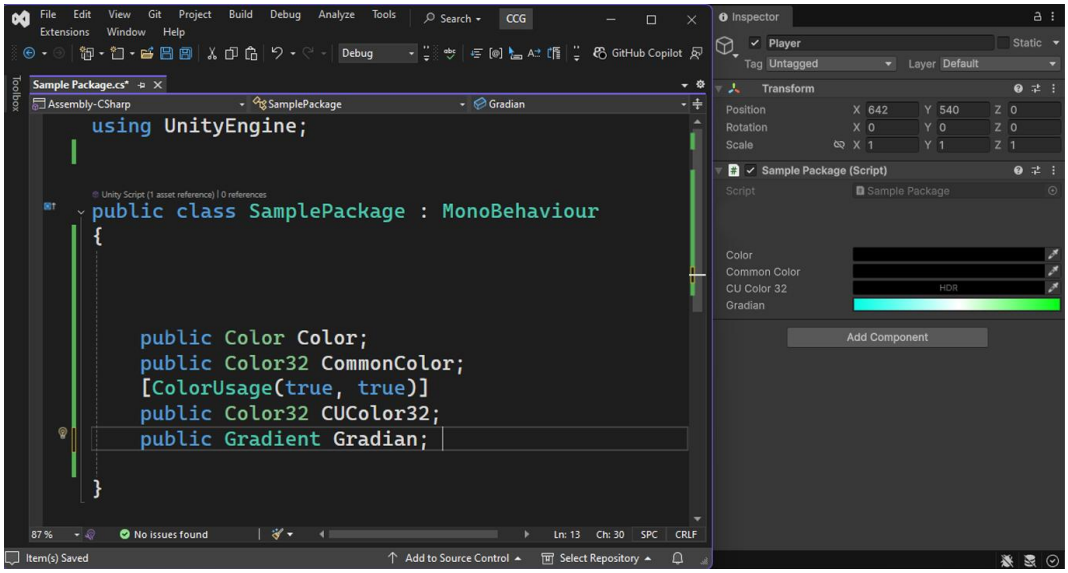
- رنگ Color

برای استفاده از رنگ‌ها مورد استفاده قرار می‌گیرد. مخصوصا برای ساخت پکیج‌های مرتبط با متریال استفاده می‌شود. همچنین برای فعال کردن حالت HDR در رنگ، می‌توان استفاده از قابلیت رنگ‌ها را فعال نمود. در رابطه ۱۰ و شکل ۱۰ طرز کار با انواع رنگ‌ها نمایش داده شده است.

- طیف Gradient

با استفاده از این ابزار می‌توان یک طیف رنگ را انتخاب نمود. از این ابزارها نیز بیشتر برای کارهای نمایشی مانند جلوه‌های تصویری و سیستم ذرات و متریال‌ها استفاده می‌شود. در رابطه ۱۰ به ترتیب رنگ ساده با استاندارد۳۲، رنگ ساده، رنگ با HDR و سپس طیف رنگی ایجاد شده است.

```
public Color32 نام;
public Color نام;
[ColorUsage(true,true)]
public Color32 نام;
public Gradient نام;                                (10)
```

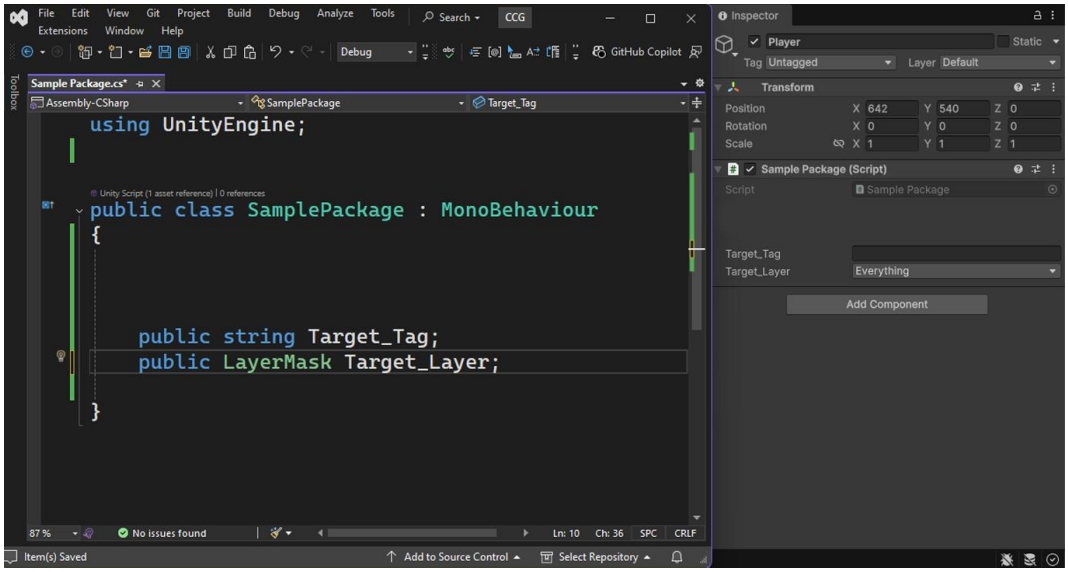


شکل ۱۰. استفاده از متغیرهای مرتبط با رنگ و طیف رنگی.

➢ لایه و برچسب (Layer & Tag)

در نرم‌افزار یونیتی می‌توان یک لایه و یک برچسب به آبجکت‌ها منتسب نمود تا آنها را دسته‌بندی کنیم. مثلا در متن بازی، تمامی دوستان را با برچسب friends و دشمنان را با برچسب enemies می‌توان دسته‌بندی نمود. این کار باعث می‌شود تا در فرآیند برنامه‌نویسی، دسترسی به آنها آسان‌تر شده و عملیات مورد نظر بر روی آنها انجام پذیرد. در رابطه ۱۱ ابتدا متغیر لایه و سپس متغیر رشته‌ای برای برچسب قرار دارد. در شکل ۱۱ استفاده از این ابزارها نمایش داده شده است.

```
public LayerMask نام دلخواه;
public String نام دلخواه;                                    (11)
```



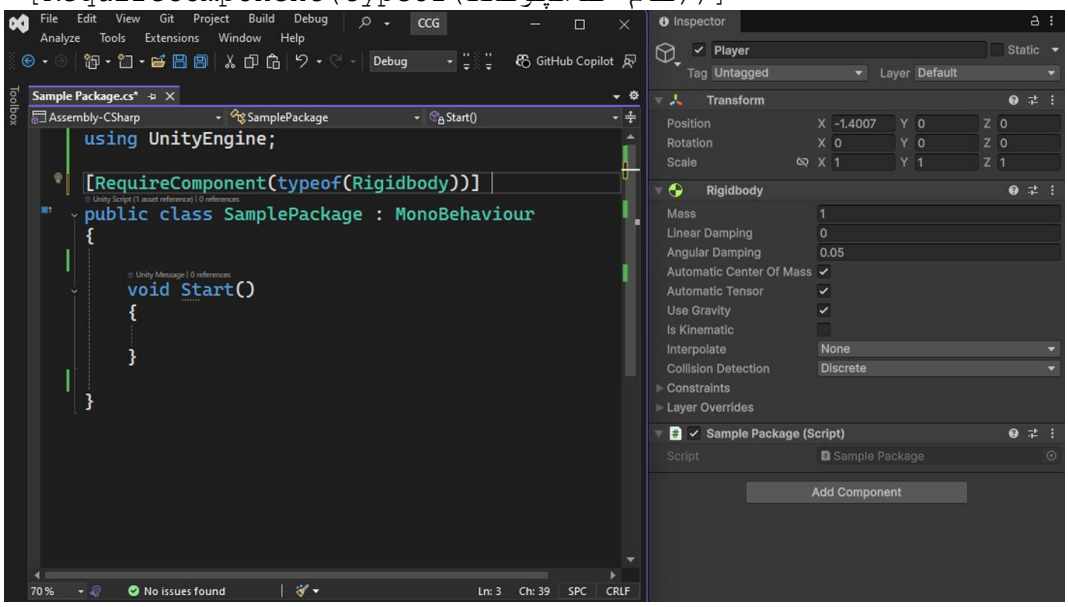
شکل ۱۱. استفاده از متغیر لایه و سپس متغیر رشته‌ای برای برچسب.

نکته: لایه، عناوینی مشخص است که به هرکدام عددی نسبت داده می‌شود و نهایتا هر بازی می‌تواند ۳۲ لایه داشته باشد. برچسب محدود به تعداد نیست و منتسب به عددی نیز نمی‌باشد. اسامی برچسب‌ها قابل تغییر است و به همین سبب از نوع رشته‌ای هستند. همانطور که در شکل ۱۱ دیدیم؛ برچسب‌ها متغیر اختصاصی خود را ندارند و باید اسم برچسب را به عنوان یک نوشته وارد نمود.

➢ اضافه کردن کامپوننت موردنظر

گاهی اوقات برای استفاده کامل از قابلیت‌های یک پکیج در محیط یونیتی، لازم است که از کامپوننت‌های یونیتی نیز استفاده شود. به عنوان مثال کامپوننت RigidBody برای کارهای فیزیکی مانند جاذبه، شتاب و برخورد و غیره مورد استفاده قرار می‌گیرد. پس اگر بخواهیم از کدهای شتاب و جاذبه و برخورد استفاده کنیم؛ مجبور به اضافه نمودن این کامپوننت هستیم. طبق رابطه ۱۲ می‌توان کامپوننت دلخواه را به اضافه کرد. در شکل ۱۲ نیز یک نمونه از آن نمایش داده شده است.

```
[RequireComponent(typeof(نام کامپونت))]                    (12)
```



شکل ۱۲. اضافه کردن کامپوننت اجباری با کدنویسی

نکته: اجازه حذف کامپوننت مورد نظر داده نمی‌شود. بسیار اهمیت دارد که چه کامپوننت‌هایی را به صورت اجباری اضافه کنیم زیرا دیگر قابلیت حذف از بین می‌رود.

نکته۲: گاها جهت اطمینان بیشتر، در تابع شروع نیز استعلام می‌گیرند که آیا کامپوننت مورد نظر وجود دارد یا خیر و اگر وجود نداشت؛ اعلام خطا می‌شود. اعلام خطا در رابطه ۱۴ صحبت آورده شده است.

نکته۳: طبق شکل ۱۲ باید اسم کامپوننت قبل از اسم کلاس مورد نظر نوشته شده باشد.

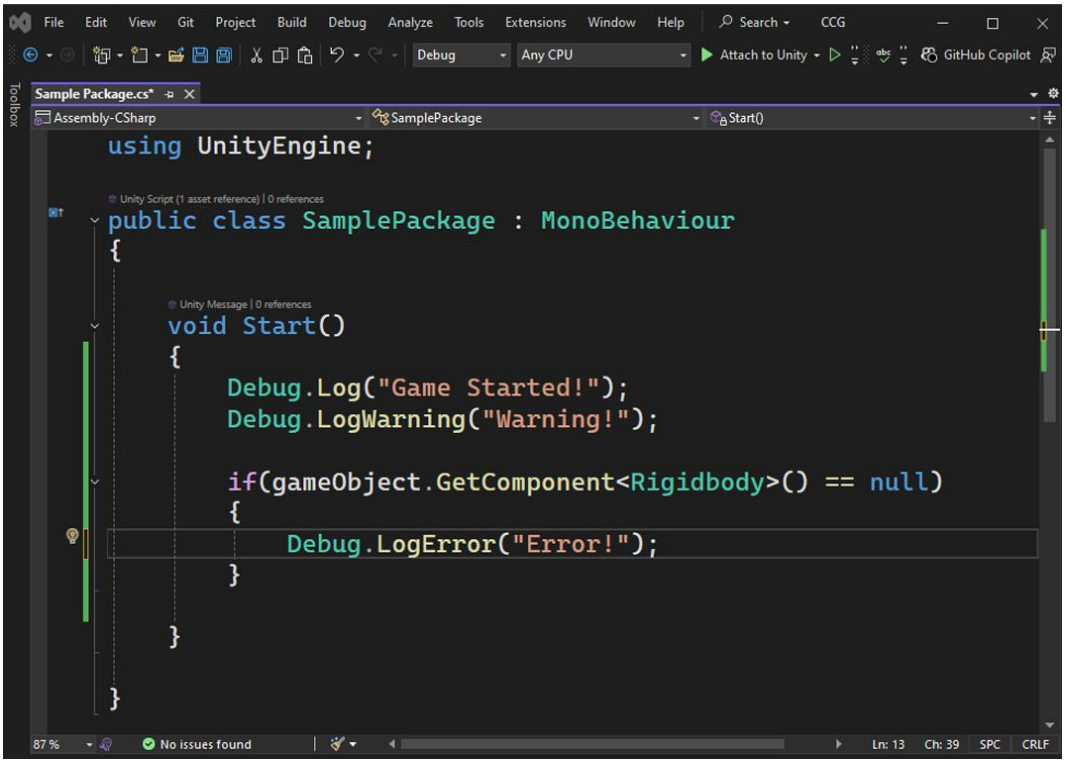
نکته۴: راه‌های دیگری هم برای اضافه کردن و یا حذف کامپوننت وجود دارد که در رابطه ۱۳ به آنها اشاره شده است. این کدها باید در تابعی اجرا شوند. خط اول برای اضافه کردن کامپوننت و خط دوم برای حذف کامپوننت می‌باشد.

```
gameObject.AddComponent<نام کامپونت>();
Destroy(gameObject.GetComponent<نام کامپونت>());          (13)
```

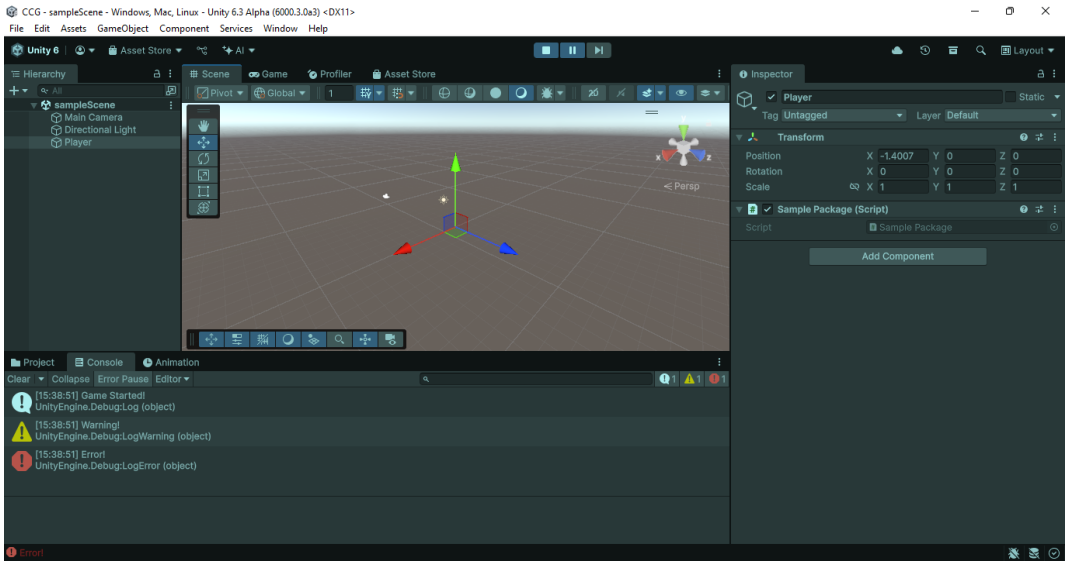
➢ پیام معمولی یا هشدار و خطا

گاهی ممکن است که نیاز به پیام هشدار یا خطا شود. معمولا اینکار برای حفظ سلامت پکیج است. به عنوان مثال اگر پکیج شرایطی را برای اجرای صحیح بخواهد و آن وجود نداشته باشد؛ هشداری می‌دهد. گاها نیز نیاز به یک کامپوننت دارد تا اجرا شود و اگر نباشد؛ کل کار اجرا نخواهد شد پس در این صورت با اعلام خطا، ادامه کار متوقف خواهد شد. در رابطه ۱۴ ابتدا یک نمونه پیام ساده و سپس هشدار و بعد از آن یک خطا نمایش داده شده است.

```
Debug.Log("پیام ساده");
Debug.LogWarning("پیام هشدار");

if(شرط مورد نظر)
{Debug.LogError("پیام خطا");}                              (14)
```



شکل ۱۳. سه نوع پیام عادی، هشدار و خطا کدنویسی شده‌اند.



شکل۱۴. در پنجره پایین، نحوه نمایش سه نوع پیام، نمایش داده شده است.

نکته: پیام‌ها باید در تابعی اجرا می‌شوند. به عنوان مثال در تابع شروع اجرا شوند که در شکل ۱۳ نمایش داده شده است.

نکته۲: این پیام‌ها در پنجره console در نرم‌افزار یونیتی نمایش داده خواهند شد که در شکل ۱۴ آورده شده است.

نکته۳: پیام خطا، بازی را متوقف می‌گرداند. پس بهتر است کمتر از آن استفاده شود یا اینکه حتما با کد شرطی مانند نمونه که در شکل ۱۳ و رابطه ۱۴ آورده شده است؛ مورد استفاده قرار گیرد.

باید توجه نمود که سازوکار debug یعنی پیام‌هایی که در رابطه ۱۴ ذکر شده‌اند؛ با ابزارهای دیگر مانند سربرگ، تفاوت‌هایی دارد. در ادامه به تفاوت این دو سازوکار اشاره خواهد شد.

ابزارهای ساخت پکیج

- ابزارها فقط در یونیتی کار می‌کنند و برای بهبود تجربه کاربری در پنجره Inspector طراحی شده‌اند.
- در ساخته نهایی کاملا حذف می‌شوند و هیچ کدی از آنها در بازی نهایی اجرا نمی‌شود.
- هیچ تاثیری روی عملکرد بازی ندارند و فقط برای سازماندهی و نمایش بهتر متغیرها در زمان کدنویسی استفاده می‌شوند. مانند ابزار سربرگ و راهنمای ابزار.

توابع پیام‌ها Debug

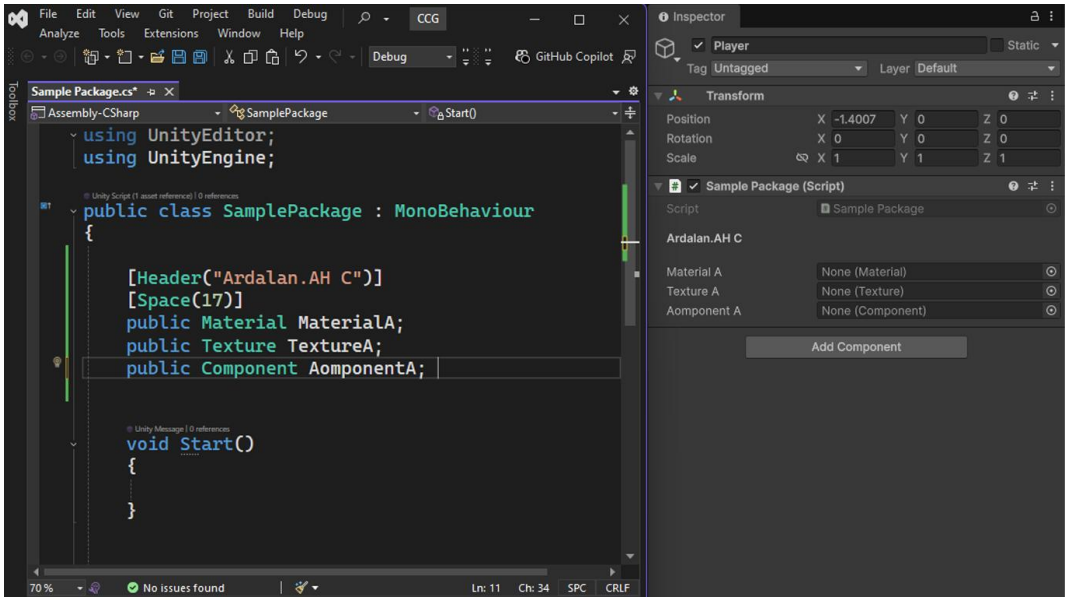
- در هر دو محیط یونیتی و ساخته نهایی اجرا می‌شوند مگر اینکه به صورت دستی آنها را مدیریت نمود.
- در ساخته نهایی هم پردازش می‌شوند و می‌توانند روی عملکرد بازی تاثیر بگذارند. به ویژه اگر در بخش‌هایی مثل توابع Update یا FixedUpdate استفاده شوند.
- برای حذف یا غیر فعال کردن آنها در ساخته نهایی، باید از کدهایی مانند (#if UNITY_EDITOR) یا روش‌های مشابه استفاده نمود.

نتیجه: پیام‌ها می‌توانند در سرعت، کیفیت و بهینه‌بودن ساخته نهایی تأثیرات منفی بگذارند. نباید از آنها زیاد استفاده نمود.

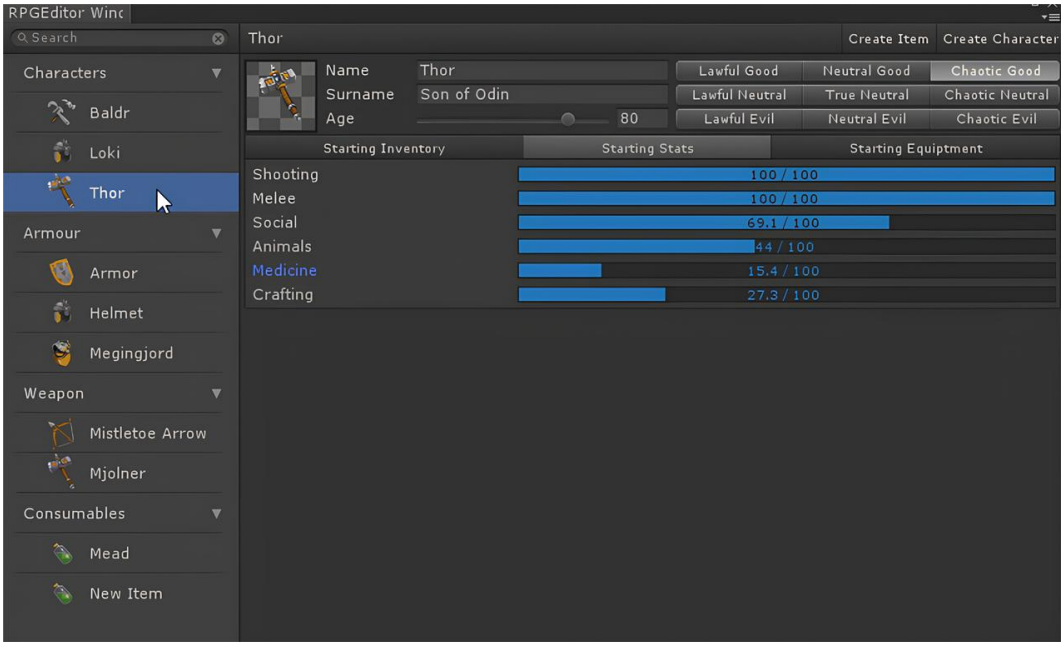
- دیگر ابزارها

ابزارهایی بسیار بیشتری نیز برای ساخت انواع پکیج وجود دارند. در شکل ۱۵ و رابطه ۱۵، بخش دیگری از متغیرها نمایش داده شده است. به ترتیب سه متغیر متریال، بافت (تکسچر) و کامپوننت ایجاد شده‌اند. هر متغیری می‌تواند در بازیسازی مهم بوده و در بخش پکیج‌نویسی نیز مورد استفاده قرار گیرد. به عنوان مثال در نسخه یونیتی ۶ و جدیدتر، ورودی‌ها متغیرهای بسیار جامع‌تری شده‌اند و استفاده از آنها اهمیت بیشتری پیدا کرده است.

همچنین در شکل ۱۶ یک تصویر از پکیجی قرار دارد که به راحتی می‌توان بازی‌ها را مدیریت نمود. این نوع پکیج‌سازی برای بازیسازان بسیار مطلوب است، اما ساخت آن پیچیده و نیازمند ابزارهایی مانند ODIN Inspector می‌باشد.



شکل ۱۵. نمایش سه متغیر جنس (متریال)، بافت و کامپوننت.



شکل ۱۶. یک نمونه پکیج برای مدیریت راحت‌تر پروژه که برای ساخت یک بازی جنگی می‌باشد.

۳.۳. تحلیل فنی و کاربردی پکیج‌های ساخته شده

در این بخش، به منظور نشان دادن کاربرد عملی مفاهیم مطرح‌شده در بخش‌های قبلی، پکیج نمونه طراحی و پیاده‌سازی شده است. هدف از ساخت این پکیج‌ها، نمایش چگونگی تسریع فرآیند توسعه بازی و بهبود تجربه کاربری با استفاده از ابزارهای آماده است. پکیج با توجه به نیازهای رایج بازیسازان انتخاب شده است.

این مراحل دقیقا طبق همان مراحلی که در نمودار ۱ صحبت شده بود؛ پیش خواهند رفت. دقت داشته باشید که یکبار به صورت ساده و مثالی، توضیح داده خواهد شد تا روند کار بهتر درک شود.

- شروع کار: بهتر است با تمرکز کارها را شروع نماییم و اگر چیزی باعث از بین رفتن تمرکز شده یا درگیر پروژه‌های قبلی هستیم؛ فعلا سمت پکیج‌سازی نرویم.
- تعریف نیاز: چند نیاز برای خود در نظر می‌گیریم. به عنوان مثال مدیریت صدا در پروژه را به عنوان یک نیاز در نظر می‌گیریم. اینکه چگونه صداها در محیط بازی کم و زیاد شوند و این سیستم باید ذخیره هم بشود. این یک نیاز رایج است و در بازیسازی ایران نیز بسیار کاربرد دارد.
- ایده‌پردازی: می‌دانیم که چنین چیزی در برنامه‌نویسی وجود داشته و وجود دارد. پس امکان ساخت چنین پکیجی میسر است. برای تغییر میزان صدا در محیط یونیتی، ابزارهای مختلفی داریم. باید ببینیم کدام ابزار بهینه‌تر است.
- پیاده‌سازی ممکن است؟ بله امکان دارد. البته که ساخت پکیج‌های خوب و بهینه سخت اما امکان‌پذیر است.
- تحلیل و بررسی: استودیوها، هر بار برای ساخت بازی، زمان زیادی را برای مدیریت صدا صرف می‌کنند و قطعا نیاز به یک پکیج خوب دارند تا بارها از آن استفاده کنند. بدین صورت از لحاظ زمان، اقتصاد و مسائل فنی، مطلوب است.
- موفق خواهد بود؟ به احتمال زیاد بله موفق خواهد بود. پس کار را ادامه می‌دهیم.
- طراحی معماری کلی پکیج: نیاز به گروه‌های صوتی هست تا در منوی تنظیمات صدا، با اسلایدر کم و زیاد شوند. عدد نهایی نیز باید ذخیره شود تا با بازکردن مجدد بازی، صداها به همان میزان تنظیم شده برگردند[5,4].
- برنامه‌نویسی و پیاده‌سازی: با آزمون و خطا، چندین بار برنامه‌نویسی نموده و اجرا می‌کنیم. (در نهایت نسخه نهایی در رابطه ۱۶ اشاره شده است.) شاید ابتدا استفاده از گزینه AudioMixer بسیار خوب و ساده بنظر آمد. این ابزار نوعی میکسر است که می‌تواند صداها را به صورت گروهی مدیریت و میکس کند. اما یک مشکل اساسی دارد: نمی‌توان

صدایی را به طور کامل قطع نمود. فقط می‌توان صدای مورد نظر را تا منفی ۸۰ دسیبل (-80 db) محدود نمود. پس برای همین به سراغ کنترل تمامی AudioSource ها و برچسب‌ها رفتیم. وگرنه پیاده‌سازی از طریق AudioMixer بسیار راحت‌تر خواهد بود اما فقط همین یک مشکل را دارد. در شکل ۱۷، نحوه آزمایش پکیج در محیط آزمایشی نمایش داده شده است. در تصویر ۱۸ نیز بخشی از کار با AudioMixer را مشاهده می‌فرمایید که در نهایت با شرح جزئیات برای استودیو هدف، روش دیگری را برای ساخت پکیج در پیش گرفتیم.

- آزمایش پکیج: انواع کدها را امتحان می‌کنیم. در نهایت پکیج به خوبی کار می‌کند و مشکل خاصی ندارد.
- آیا بهینه است؟ چندین بار پیاده‌سازی نمودیم تا بهینه‌ترین کد را پیدا کنیم. پس بهینه است.
- پیاده‌سازی شمایل پکیج: با استفاده از ابزارها مانند سربرگ و راهنمای ابزار و غیره، یک شمایل خوب برایش نمایه‌سازی می‌کنیم. باید در این مرحله، دقت زیادی در پیاده‌سازیس به عمل آورد. باید طوری پکیج را ایجاد نمود که بازیسازان با آن ارتباط برقرار کنند. نباید گیج‌کننده باشد و یا دارای نکات انحرافی و اضافی باشد. هرچه کار با پکیجی راحت‌تر و بهینه‌تر باشد؛ مطلوب‌تر است و این امر، علم و تجربه پکیج‌ساز را نمایش می‌دهد.
- مستندسازی: آموزش کار با پکیج و نسخه‌های سازگار با آنرا در قالب محتوای متنی یا همراه با عکس و ویدیو تهیه می‌کنیم تا موقع استفاده از پکیج، بازیسازان سردرگم نشوند. بدین صورت پیاده‌سازی پکیج ممکن می‌شود. به عنوان مثال این یک نمونه راهنما و مستندسازی برای پکیجی است که در رابطه ۱۶ و ۱۷ ذکر شده است:

این پکیج با هدف تنظیم صداهای بازی طراحی شده است. سازنده آن «استودیو فلان» می‌باشد و برای پشتیبانی نیز می‌توانید به سایت استودیو مراجعه کنید. در ادامه آموزش کار با پکیج قرار دارد.

این پکیج دارای دو کد اصلی به نام‌های کد مرجع و کد زیرمجموعه است. کد مرجع برای کنترل تمامی زیرمجموعه‌ها در یک گروه است و تمامی آن زیرمجموعه‌ها باید دارای کد زیرمجموعه باشند. کد مرجع چهار ورودی لازم دارد که به همراه تنظیمات آن در ادامه ذکر می‌شود:

از شما یک اسلایدر می‌خواهد. این اسلایدر volumeSlider نام دارد که باید اسلایدر کم و زیاد کردن صدای مورد نظر را به آن نسبت دهید. هر کد مرجع برای یک گروه است پس به تعداد گروه‌هایی که استفاده می‌کنید؛ به همان تعداد اسلایدر نیز نیاز می‌باشد. در تنظیمات اسلایدر باید تابع مرتبط با این کد یعنی OnVolumeChanged داده شود.

یک متغیر از نوع رشته‌ای (متنی) به نام volumeText وجود دارد که عدد درصد بلندی صدا را نمایش دهد. این عدد باید زیرمجموعه اسلایدر باشد تا با آن حرکت کند. دقیقا همانند شکل ۱۷ عدد نمایش داده خواهد شد. این متغیر باید از نوع TMP_Text باشد. البته اگر این مورد خالی بماند نیز مشکلی پیش نمی‌آید اما به عنوان تزئین زیباست. مکان و رنگ عدد دست خود طراح بازی است.

برچسبی با عنوان audioTag وجود دارد. تمامی آبجکت‌هایی زیر مجموعه باید دارای یک برچسب (تگ) باشند. این برچسب دقیقا باید در اینجا وارد شود. اگر به درستی وارد نشود؛ صدای آبجکت‌ها در وسط بازی قابل تغییر نیست.

یک متعیر رشته‌ای دیگر با نام volumeKey وجود دارد. این نام باید با نام دیگر مجموعه‌ها تفاوت داشته باشد. این اسمی است که میزان صداها را در فایل‌های بازی ذخیره می‌سازد. هر نامی می‌توان برای آن وارد نمود اما هر مجموعه باید نام خودش را برای ذخیره داشته باشد.

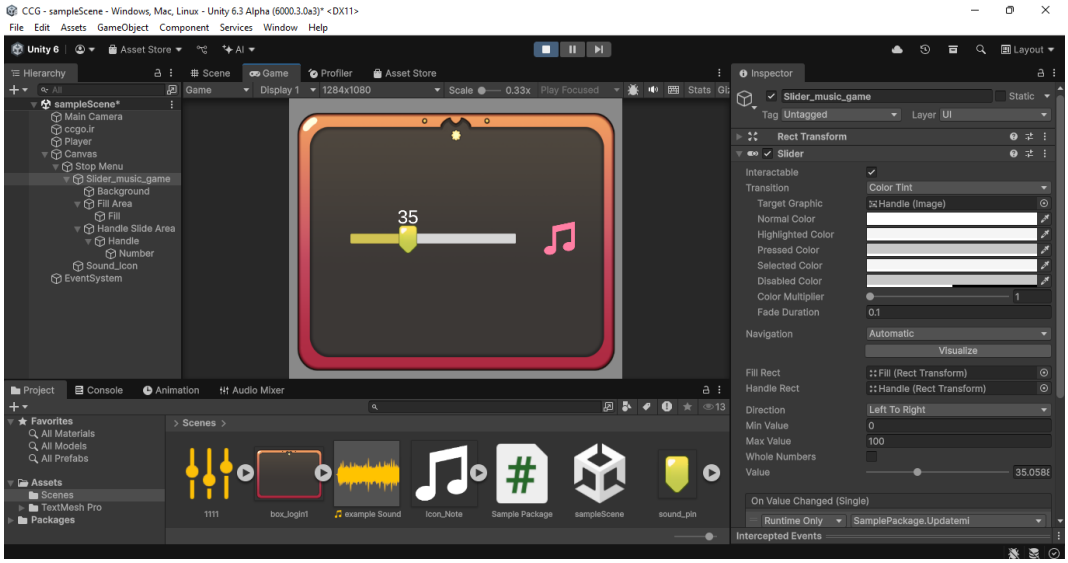
ورودی‌ها و تنظیمات کد زیرمجموعه‌ها: این کد باید به تمامی زیرمجموعه‌ها داده شود.

متغیری از آبجکت‌ها دارد که thisSoundManager نام دارد. در این بخش باید آبجکت مرجع مورد نظر داده شود. بدین صورت تمامی زیرمجموعه‌ها در یک مجموعه و به سرپرستی مرجع مورد نظر قرار می‌گیرند.

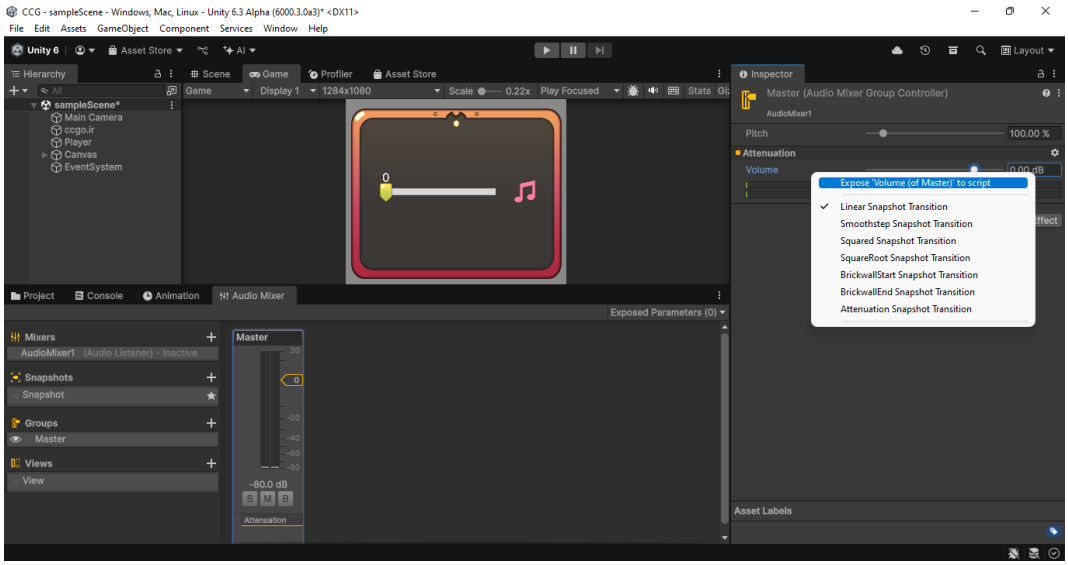
با تنظیم ورودی‌ها، پکیج کار می‌کند و نیاز به کار دیگری نیست.

- آماده‌سازی نهایی: اگر نیاز به فایل یا هماهنگی خاصی هست؛ انجام می‌پذیرد.

- استفاده/ انتشار: در استودیو خودمان مورد استفاده قرار می‌گیرد./ همچنین کوک (۲۰۲۱) در بخشی از کتاب خود یعنی «Package Distribution and Publishing» به اهمیت مستندسازی و پشتیبانی فعال در فرآیند انتشار پکیج‌ها در Asset Store اشاره می‌کند[6].
- پشتیبانی و بروزرسانی: اگر کار نیاز به بهبود و اصلاح داشته باشد؛ به ما می‌گویند تا آنرا بروزرسانی کنیم.
- پایان: کار به پایان رسید و پکیج با موفقیت ساخته و منتشر شد.



شکل ۱۷. پیاده‌سازی اولیه پکیج در یک محیط آزمایشی.



شکل ۱۸. بخشی از تلاش‌ها برای کار با ابزارهای مختلف در جهت ساخت پکیج

در نهایت یک برنامه بهینه برای این امر ارائه شد که در رابطه ۱۶ و ۱۷ آنرا مشاهده می‌فرمایید. گاها ممکن است که یک پکیج شامل چند کد مختلف باشد و در این پکیج نیز برای بهینه‌سازی این پکیج دارای دو کد است.

این پکیج در نسخه یونیتی ۲۰۲۳ و یونیتی ۶.۳ (۲۰۲۵) در سال ۱۴۰۴.ه.ش برابر با ۲۰۲۵.م آزمایش شده و کاملا صحیح کار کرده است. نحوه نمایش رابطه ۱۶ و ۱۷ داخل محیط یونیتی، در شکل ۱۹ نمایش داده شده است.

کد مرجع:

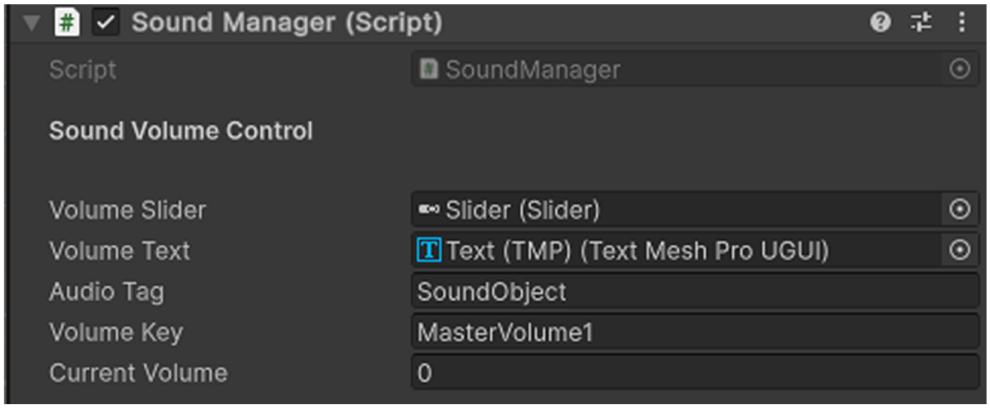
```
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using System.Collections.Generic;

public class SoundManager : MonoBehaviour
{
    [Header("Sound Volume Control")]
    [Space(21)]
    [Tooltip("Slider for adjusting the master volume level (0-
100).")]
    public Slider volumeSlider;
    [Tooltip("TextMeshPro UI element to display the current volume
percentage (0-100).")]
    public TMP_Text volumeText;
    [Tooltip("Tag used to identify GameObjects with AudioSources
that should be controlled by this manager.")]
    public string audioTag = "SoundObject";
    [Tooltip("PlayerPrefs key used to save and load the master
volume setting.")]
    public string volumeKey = "MasterVolume";

    public float currentVolume;
    private List<AudioSource>audioSources = new List<AudioSource>();

    void Awake()
    {
        if (volumeSlider != null)
        {
            currentVolume = PlayerPrefs.GetFloat(volumeKey, 1f);
            volumeSlider.minValue = 0;
            volumeSlider.maxValue = 100;
            volumeSlider.value = (int)(currentVolume * 100);
        }
        if (volumeText != null)
            volumeText.text =
Mathf.RoundToInt(volumeSlider.value).ToString();

        UpdateAllAudioVolumes();
    }

    public void OnVolumeChanged()
    {
        if (volumeSlider == null || volumeText == null)
            return;
        currentVolume = volumeSlider.value / 100f;
        volumeText.text = Mathf.RoundToInt(currentVolume *
100).ToString();
        PlayerPrefs.SetFloat(volumeKey, currentVolume);
        UpdateAllAudioVolumes();
    }
```

```
    private void UpdateAllAudioVolumes()
    {
        GameObject[] taggedObjects =
GameObject.FindGameObjectsWithTag(audioTag);
        foreach (GameObject obj in taggedObjects)
        {
            AudioSource source = obj.GetComponent<AudioSource>();
            if (source != null)
            {
                source.volume = currentVolume;
            }
        }
    }
}                                                                    (16)
```

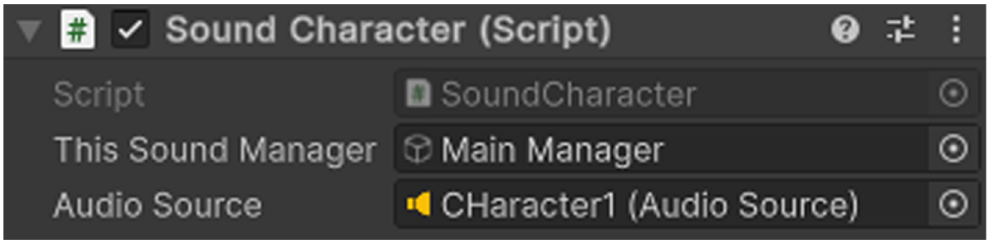
کد زیرمجموعه‌ها:

```
using UnityEngine;
[RequireComponent(typeof(AudioSource))]
public class SoundCharacter : MonoBehaviour
{
    [Tooltip("Reference to the SoundManager GameObject in the
scene.")]
    public GameObject thisSoundManager;
    [SerializeField]
    private AudioSource audioSource;
    void Awake()
{
        audioSource = GetComponent<AudioSource;()<
        calculate;()
}
    void OnEnable()
    {
        calculate;()
    }
    void calculate()
    {
        if (thisSoundManager == null ||
thisSoundManager.GetComponent<SoundManager>() == null)
            return;
        audioSource.volume =
thisSoundManager.GetComponent<SoundManager>().currentVolume;
    }                                                                (17)
```

نحوه نمایش پکیج در محیط یونیتی
کد مرجع برای مقداردهی صدا



نحوه نمایش کد زیرمجموعه‌ها



شکل ۱۹. نحوه نمایش پکیج ساخته شده.

ساخت پکیج زمان‌بر است. زیرا باید خوب ایده‌پردازی شود. بارها مورد آزمایش قرار گیرد. مستندسازی شود. به کارفرما و یا خریداران، نمایش داده شود تا مورد پسند آنان نیز قرار گیرد. ممکن است کارفرما ابزارهای استفاده شده را محدود کند و این مهم، کار را سخت‌تر نیز می‌کند. همچنین مشکلات دیگری نیز ممکن است در اثنای کار رخ دهد. همه اینها از سختی‌های کار پکیج سازی است و شخصی که پکیج می‌سازد؛ حتما باید آنرا چندین دفعه و با چندین حالت مختلف امتحان کند تا بهترین نتیجه را بگیرد. ممکن است از یک پکیج، سالها استفاده شود و حتی ریزترین ایرادات، باعث می‌شود تا کیفیت کار با آن پایین بیاید. پکیج‌ساز یکبار باید زمان و دقت فراوانی را خرج ساخت پکیج کند تا استفاده‌کنندگان همیشه بتوانند به سهولت و اطمینان از آن استفاده کنند و این مورد پسند تمام پکیج‌سازان است.

در این بخش سعی شد یک نمونه پکیج را با استفاده از همان ابزارها ساخته و به نمایش درآوریم. مراحل و چالش‌های تولید را بررسی کنیم و در نهایت، با موفقیت کدی نوشته شده است که می‌توان بارها و در بازی‌های مختلف از آن استفاده نمود.

۴.۳. چالش‌ها و محدودیت‌های فنی، اقتصادی و رقابتی

ساخت پکیج همیشه با چالش‌ها و محدودیت‌های زیادی مواجه است. در ادامه به بزرگترین چالش‌ها اشاره شده است.

- محدودیت‌های فنی

یکی از اصلی‌ترین چالش‌های ساخت پکیج‌های آماده، محدودیت‌های فنی و سازگاری با نسخه‌های مختلف یونیتی است. ممکن است پکیجی که برای یک نسخه خاص از یونیتی طراحی شده؛ در نسخه‌های جدیدتر یا قدیمی‌تر دچار مشکل شود و یا نیاز به بروزرسانی‌های مداوم داشته باشد. همچنین، برخی پکیج‌ها ممکن است با دیگر پکیج‌های محبوب ناسازگار باشند و باعث بروز خطا یا کاهش عملکرد بازی شوند. بهینه‌سازی کدها و مدیریت حافظه نیز از جمله چالش‌های فنی است که باید به دقت مورد توجه قرار گیرد، زیرا پکیج‌های سنگین می‌توانند سرعت و کیفیت بازی نهایی را تحت تأثیر قرار دهند. مثلا یکی از مواردی که

می‌تواند پکیج را سنگین کند؛ بررسی کدها در تابع Update است. زیرا این تابع در هر فریم (قاب نمایشی) یکبار اجرا می‌شود. یعنی هر ثانیه، ممکن است ۳۰ یا ۶۰ دفعه اجرا شود و این مطلوب نیست. باید کدها طوری تنظیم شوند که فقط موقع فراخوانی اجرا و بررسی شوند و نیازی نیست که در هر فریم بررسی و اجرا شوند. به طور مثال هر چندثانیه یکبار که نیاز به محاسبه می‌باشد؛ فراخوانی و بررسی شوند.

- محدودیت‌های اقتصادی

ساخت و فروش پکیج‌های آماده در Asset Store یا دیگر پلتفرم‌ها، نیازمند سرمایه‌گذاری اولیه برای توسعه، تست، مستندسازی و بازاریابی است. بسیاری از بازیسازان مستقل یا استودیوهای کوچک ممکن است توان مالی کافی برای تولید پکیج‌های باکیفیت و رقابت با محصولات موجود را نداشته باشند. همچنین، قیمت‌گذاری مناسب و جذب مشتری در بازار رقابتی، نیازمند استراتژی‌های بازاریابی هوشمندانه و شناخت دقیق نیازهای بازار است. در برخی موارد، ممکن است پکیج ساخته شده به دلیل عدم استقبال بازار یا وجود رقبای قوی، به فروش نرسد و سرمایه‌گذاری انجام شده به هدر برود.

- محدودیت‌های رقابتی

بازار پکیج‌های آماده یونیتی بسیار رقابتی است و هر روزه پکیج‌های جدیدی با ویژگی‌های پیشرفته‌تر و قیمت‌های پایین‌تر وارد بازار می‌شوند. این موضوع باعث می‌شود تا بازیسازان برای ماندن در بازار، مجبور به نوآوری مداوم و ارائه خدمات پشتیبانی بهتر باشند. همچنین، برخی استودیوهای بزرگ با تولید پکیج‌های جامع و چندمنظوره، بازار را اشباع کرده و فرصت رشد برای پکیج‌های کوچک و تخصصی را کاهش می‌دهند. در چنین شرایطی، تمایز محصول و ارائه ویژگی‌های منحصر به فرد، کلید موفقیت است.

- تأثیر هوش مصنوعی AI

با پیشرفت هوش مصنوعی، ابزارهای جدیدی برای تولید کدها و پکیج‌ها در حال ظهور هستند. این موضوع می‌تواند هم فرصت و هم چالش باشد. از یک سو، هوش مصنوعی می‌تواند فرآیند ساخت پکیج را تسریع کند اما از سوی دیگر، ممکن است باعث کاهش تقاضا برای پکیج‌های دستی و افزایش رقابت شود. بازیسازان و پکیج‌سازان باید با استفاده از ابزارهای هوش‌مصنوعی، کیفیت و سرعت تولید خود را افزایش دهند تا بتوانند در بازار رقابتی باقی بمانند. هوش مصنوعی قدرتمند شده است و می‌تواند پکیج بسازد اما بدلیل پیچیدگی پکیج‌ها، ممکن است اشتباهاتی نیز در این امر داشته باشد که همچنان باید ایرادات آنها به صورت دستی گرفته شود و نسخه بهینه‌تری ارائه شود.

جالب است که برخی پکیج‌ها، خود دارای قابلیت‌های هوش مصنوعی هستند. یعنی بازی را بر اساس یک هوش، تحلیل و یا کدهایی را اجرا می‌کنند. می‌توانند یک دشمن بسیار سرسخت باشند و یا یک دوست وفادار که تا پای جان با دشمنان مبارزه کند. همچنین برخی برای سوال و جواب و یا راهنمای بازی و حتی ترجمه صحبت‌ها به زبان‌های مختلف، از پکیج‌های دارای هوش‌مصنوعی استفاده می‌کنند. از این رو یادگرفتن زبان‌های هوش‌مصنوعی و دیگر زبان‌های برنامه‌نویسی مانند C++ یا پایتون برای ساخت پکیج در یونیتی، پیشنهاد می‌شود.

این موارد، بزرگترین و اساسی‌ترین چالش‌های ساخت پکیج هستند که در این بخش معرفی شدند. مدت‌زمان ساخت پکیج به نسبت دیگر کارها در استودیوها کوتاه‌تر است و صرفا یکبار باید ساخته شوند و تا سالیان متمادی، از همان پکیج‌ها استفاده می‌شود. طبیعی است که درآمد پکیج‌سازی باید بیشتر از دیگر کارها باشد تا درآمد این گروه حفظ شود.

۴. نتیجه‌گیری

این پژوهش با هدف بررسی نقش پکیج‌های کدنویسی آماده در موتور بازی‌سازی یونیتی انجام شد. این پکیج‌ها تأثیر قابل توجهی بر تسریع فرآیند ساخت بازی، کاهش هزینه‌ها و افزایش بهره‌وری دارند. یافته‌ها نشان داد که استفاده از این پکیج‌ها، با ارائه ابزارها و کدهای از پیش نوشته شده؛ امکان پیاده‌سازی سریع ویژگی‌های پیچیده را فراهم می‌سازد و بازیسازان را از

نوشتن کدهای تکراری بی‌نیاز می‌سازد. این امر نه تنها زمان تولید را به‌طور قابل‌توجهی کاهش می‌دهد بلکه هزینه‌های توسعه را نیز بهینه می‌سازد و امکان تمرکز بیشتر بر جنبه‌های خلاقانه و هنری بازی را فراهم می‌آورد.

این پژوهش با تحلیل دقیق معماری، مراحل ساخت و چالش‌های پیش روی پکیج‌سازان، چارچوبی عملی برای بازیسازان و استودیوهای بازی‌سازی ارائه می‌دهد. همچنین، با معرفی نمونه‌های موفق و بررسی کاربردهای عملی، نشان داده شد که پکیج‌های آماده می‌توانند به‌عنوان ابزاری مؤثر برای افزایش سرعت، کیفیت و بهره‌وری در فرآیند ساخت بازی مورد استفاده قرار گیرند. همچنین این تحقیق با توجه به کمبود منابع داخلی در این زمینه، گامی در جهت پر کردن این خلأ و ارائه راهکارهای عملی برای جامعه بازیسازان ایرانی برداشته است.

نکته حائز اهمیت این است که این مقاله، به عنوان اولین مقالات بازیسازی در ایران و اولین مقاله کار با پکیج‌های یونیتی در ایران است. از این رو مجبور به قرار دادن توضیحات فنی و معرفی ابزارها بودیم و این خودش چالش بزرگی بود. امید که روزی در این زمینه بسیار پیشرفت کنیم و منابع زیادی در دسترس داشته باشیم تا مقالات علمی متنوع‌تری را نگارش نماییم

۵. قدردانی

با تشکر و قدردانی بسیار فراوان از آقا امام زمان (عج) که همیشه ما را یاری نموده‌اند و بهترین یاور مظلومان عالم هستند.

۶. منابع و مراجع

[1] اردلان، امیرحسین. (۱۴۰۴). «فرآیند بازی‌سازی رایانه‌ای: از ایده تا انتشار؛ راهنمای جامع برای علاقه‌مندان به هنر و صنعت بازی‌سازی»، فصلنامه علمی تخصصی فناوری‌های نوین در مهندسی برق و کامپیوتر، دوره ۵، شماره ۲ (تابستان)، صص. ۶۲-۷۲.

[2] Hocking, J. (2015). Unity in Action: Multiplatform Game Development in C# with Unity 5. Manning Publications, Shelter Island, NY.

[3] Kok, B. (2021). Beginning Unity Editor Scripting. Apress, New York, NY.

[4] Hardman, C. (2024). Game Programming with Unity and C#. Apress, Berkeley, CA.

[5] Coggan, A. (2021). Unity Game Audio Implementation. Routledge, New York, NY.

[6] Kok, B. (2021). "Package Distribution and Publishing," in Beginning Unity Editor Scripting. Apress, New York, NY.

[7] Hardman, C. (2024). "Menus and UI," in Game Programming with Unity and C#. Apress, Berkeley, CA.

[8] Hardman, C. (2024). "Build Mode UI," in Game Programming with Unity and C#. Apress, Berkeley, CA.

Accelerating Game Development: Leveraging Pre-built Code Packages in the Unity Engine

Amirhossein Ardalan*
Game Design Instructor, Avicenna International College, Tehran, Iran; Tbilisi, Georgia.
(shsbbhorse@chmail.ir)

Abstract— The process of computer game development is complex and multi-stage, requiring collaboration among specialized teams across various disciplines. Programming, as one of the fundamental and time-consuming stages, plays a crucial role in the speed and quality of game production. Utilizing pre-built code packages in the Unity game engine significantly reduces programming time, enabling developers to focus on innovation and enhancing user experience rather than writing repetitive code. This paper examines the role and benefits of these packages and analyzes examples of their application in different projects. The adoption of this approach is common not only in large studios but also in small teams, facilitating better time management, greater focus on narrative and artistic aspects, and increased productivity. This research is based on practical experiences and references to the teachings of prominent experts in Unity package development, providing a detailed exploration of various aspects of these packages, including their technical capabilities and economic advantages.

Keywords: Game Development, Unity, Unity Engine, Programming, Computer Game, Productivity, Package Development, AI