گزارش پروژه بازی بهینه‌سازی‌شده با اسمبلی

ساختار و زبان کامپیوتر

کیارش صانعی

دکتر امیرحسین جهانگیر

بهمن ۱۴۰۳

۱. مقدمه

در این پروژه، هدف اصلی بهینه‌سازی بخش‌های محاسباتی سنگین یک بازی با استفاده از زبان اسمبلی است. با توجه به اینکه برخی از محاسبات بازی در هر فریم اجرا می‌شوند، نیاز به بهینه‌سازی این بخش‌ها احساس شد. استفاده از اسمبلی باعث افزایش سرعت پردازش شده و نرخ فریم بازی (FPS) را بهبود می‌بخشد.

۲. ساختار پروژه

پروژه شامل دو بخش اصلی است:

- **کدهای اسمبلی (*ASM/*)**: شامل توابعی که برای بهینه‌سازی مسیر حرکت توپ، جزئیات رسم توپ، گرادیان‌ها و مسیرهای مختلف مورد استفاده قرار گرفته‌اند.
- **کدهای سی‌پلاس‌پلاس (*CPP/*)**: شامل کلاس‌ها، توابع اصلی و فایل اصلی بازی.

۲.۱. فایل‌های اسمبلی

- C.s: محاسبه مسیر منحنی توپ
- EA.s, EX.s, EY.s, SA.s, SE.s: محاسبات مربوط به رسم جزئیات توپ
- G.s: محاسبه گرادیان (Gradient)
- R.s: محاسبه مسیر عادی توپ
- S.s: محاسبه مسیر سینوسی توپ

۲.۲. فایل‌های سی‌پلاس‌پلاس

- class.cpp: شامل تمام کلاس‌های مورد نیاز بازی
- function.cpp: شامل توابع اصلی بازی
- game.cpp: فایل اصلی اجرای بازی
- kgh.h: فایل هدر پروژه

۲.۳. سایر فایل‌ها

- .gitignore: فایل مربوط به نادیده گرفتن برخی فایل‌ها در گیت
- Conclusion.md: گزارش نهایی پروژه
- game.sh: اسکریپت اجرای بازی
- log.old.txt: لاگ اجرای نسخه‌های قدیمی بازی
- log.txt: لاگ اجرای بازی‌های جدید
- README.md: مستندات پروژه

۳. ابزارهای مورد استفاده

- **زبان‌های برنامه‌نویسی**: سی‌پلاس‌پلاس، اسمبلی (x86)
- **کتابخانه‌ها**: Raylib برای پردازش گرافیکی بازی
- **کامپایلر**: G++ و NASM برای تبدیل کد اسمبلی و سی‌پلاس‌پلاس به فایل اجرایی

۴. ویژگی‌های بازی

بازی دارای دو حالت **تک‌نفره** و **چندنفره** است و سه مسیر متفاوت برای حرکت توپ دارد. همچنین سه سطح دشواری مختلف و گزینه‌هایی برای تنظیم نحوه محاسبه بخش‌های پرمحاسبه (hot parts) ارائه می‌شود.

۵. دلیل استفاده از اسمبلی

در برخی بخش‌های بازی که در هر فریم اجرا می‌شوند، پردازش زمان‌بر است. بنابراین، این بخش‌های پرتکرار (hot parts) به زبان اسمبلی پیاده‌سازی شدند تا عملکرد بازی بهبود یابد و کاربر تجربه بهتری داشته باشد. در ضمن کل بازی به زبان اسمبلی نیست به این دلیل که ابزارهای مدرن برای این به‌وجود آمده‌اند که ما از صفر همه چیز را پیاده‌سازی نکنیم!

۶. پیاده‌سازی اسمبلی در پروژه

۶.۱. نحوه استفاده از اسمبلی

تابع‌هایی که نیاز به بهینه‌سازی دارند، در فایل‌های هدر با extern "C" فراخوانی شده‌اند. این کار باعث می‌شود که این توابع با قرارداد **C calling convention** اجرا شوند.

۶.۲. فرایند کامپایل و اجرا

- از **فلگ O0-** برای غیرفعال‌سازی بهینه‌سازی‌های پیش‌فرض کامپایلر استفاده شده است.
- هنگام اجرای game.sh:
  1. کدهای اسمبلی به فایل‌های آبجکت تبدیل می‌شوند.
  2. این فایل‌ها به همراه کد سی‌پلاس‌پلاس لینک شده و بازی اجرا می‌شود.
  3. پس از بستن بازی، اطلاعات مربوط به اجرای بازی در log.txt ذخیره می‌شود.

۷. چرا اسمبلی در بخش‌های پرمحاسبه سریع‌تر است؟

۷.۱. مدیریت مستقیم رجیسترها

یکی از دلایل اصلی سرعت بالای اسمبلی این است که مستقیماً به رجیسترهای CPU دسترسی دارد. در زبان‌های سطح بالا مانند سی‌پلاس‌پلاس، متغیرها در حافظه ذخیره شده و پردازش می‌شوند، اما در اسمبلی، مقادیر مهم مستقیماً در رجیسترها ذخیره شده و پردازش می‌شوند که باعث کاهش زمان دسترسی به داده‌ها می‌شود.

۷.۲. حذف هزینه‌های اضافی کامپایلر

کامپایلرهای مدرن معمولاً بهینه‌سازی‌های مختلفی را انجام می‌دهند، اما همچنان محدودیت‌هایی دارند. استفاده از اسمبلی باعث می‌شود که بسیاری از هزینه‌های اضافی مانند **فراخوانی توابع غیرضروری، مدیریت پشته اضافی، و بارگذاری غیر موثر حافظه** حذف شوند.

۷.۳. اجرای دستورالعمل‌های اختصاصی پردازنده

برخی از عملیات خاص مانند **محاسبات برداری، عملیات‌های ممیز شناور، و پردازش همزمان چندین مقدار (SIMD)** در سطح اسمبلی کارایی بسیار بالاتری دارند. برای مثال، در S.s از دستور fsin در واحد پردازش ممیز شناور (FPU) استفاده شده است که بسیار سریع‌تر از محاسبه توابع سینوسی در سی‌پلاس‌پلاس است.

۸. نتایج و تحلیل عملکرد

۸.۱. مسیر عادی (Regular Path)

طبق log.old.txt، استفاده از اسمبلی باعث بهبود بسیار کمی سرعت محاسبات در مسیر عادی شده و به نظر می‌رسد به دلیل جنس محاسبات این بخش عملا اسمبلی کمک خاصی نکرده است.

```
Execution time is 129 seconds.
Game time is 120 seconds.
Calculation time while using C++ in mode regular is 1.000 seconds.

Execution time is 129 seconds.
Game time is 120 seconds.
Calculation time while using ASSEMBLY in mode regular is 0.999 seconds.
```

۸.۲. مسیر سینوسی (Sinusoidal Path)

در مسیر سینوسی نیز کاهش چشمگیری در زمان محاسبات مشاهده می‌شود. این به دلیل استفاده از واحد ممیز شناور (FPU) و دستور fsin در S.s است که محاسبات سینوسی را تسریع می‌کند.

```
Execution time is 131 seconds.
Game time is 125 seconds.
Calculation time while using C++ in mode sin is 12.021 seconds.

Execution time is 138 seconds.
Game time is 126 seconds.
Calculation time while using ASSEMBLY in mode sin is 9.022 seconds.
```

۸.۳. مسیر منحنی (Curve Path)

در این مسیر، بهینه‌سازی محاسبات پیچیده بدون استفاده از یک واحد اختصاصی مانند FPU انجام شده و مشاهده می‌شود که اجرای کد اسمبلی باعث افزایش سرعت پردازش شده است. به دلیل جنس محاسبات در این قسمت هم کاهش زمان خیلی زیاد نیست.

```
Execution time is 132 seconds.
Game time is 120 seconds.
Calculation time while using C++ in mode curve is 6.019 seconds.

Execution time is 131 seconds.
Game time is 120 seconds.
Calculation time while using ASSEMBLY in mode curve is 5.921 seconds.
```

۹. نتیجه‌گیری

استفاده از اسمبلی همیشه بهینه‌ترین روش نیست، اما در مواردی که **محاسبات سنگین و پرتکرار** انجام می‌شود، استفاده از آن می‌تواند باعث بهبود عملکرد برنامه و تجربه کاربری (افزایش FPS) شود. در این پروژه، با بهینه‌سازی بخش‌های حیاتی، بهبود قابل‌توجهی در سرعت بازی مشاهده شد که نشان‌دهنده موفقیت این روش است. با مدیریت دقیق رجیسترها، کاهش هزینه‌های پردازشی کامپایلر، و بهره‌گیری از قابلیت‌های سخت‌افزاری پردازنده، می‌توان به عملکردی بسیار بهینه دست یافت که در پروژه‌های مرتبط با بازی‌سازی اهمیت ویژه‌ای دارد.

دقت کنید که با توجه به نتایج استفاده از واحد ممیز شناور بیشترین تاثیر را در زمان محاسبه داشت.