Desenvolvimento de um Escalonador de Cenas para motores de jogos

Lucas Pinheiro Otaviano Andre[1]

[1]Instituto Metrópole Digital – Universidade Federal do Rio Grande do Norte (UFRN)

lpoandre@inf.ufrgs.br

Abstract. *During development of a game engine it is always important to make a well-defined separation between its game engine core and the game code. A way to meet that need is with the Scene Scheduler, where the classic Scheduler concept used by Operating Systems is applied to the Scene handling. The Scheduler has the purpose of handling each game scene and providing the necessary resources to them. Thus, making the engine internals transparent to the game developer.*

Resumo. *Durante o desenvolvimento de um motor de jogos é importante manter o código do motor separado do código do jogo. Uma maneira de obter essa separação é com um Escalonador de Cenas, onde o conceito clássico de escalonador usado em Sistemas Operacionais é aplicado ao gerenciamento de cenas. Cada cena do jogo vai ser gerenciada pelo escalonador, que deve providenciar todos os recursos e funcionalidades necessárias para a cena atual. Desta maneira, fazendo com que o código interno do motor seja transparente para os desenvolvedores de jogos.*

1. Introdução

Normalmente durante o desenvolvimento de um motor de jogos é preferível manter o código do motor separado do código do jogo. A única interação entre o código do jogo com o motor de jogos é o conjunto de funcionalidades disponíveis para o desenvolvedor, por exemplo: renderização, redes, física, informações de entradas (como teclado e controle). Desta maneira, o desenvolvedor não precisa interagir diretamente com as funcionalidades internas do motor, mas com uma interface de programação que fornece acesso aos recursos (renderização, física, redes) para o desenvolvimento do jogo.

Dessa forma, neste trabalho é apresentado o Escalonador de Cenas, um escalonador que gerência as funcionalidades e recursos fornecidos para a cena atual do jogo. Além disso, ele gerencia a memória utilizada, isto é, se a cena não está sendo mais utilizada, o escalonador vai liberar os recursos utilizados por ela, garantindo assim robustez e estabilidade ao motor de jogos. Essa técnica foi desenvolvida durante o desenvolvimento da *Nightmare Fiction Framework (NF Framework)*, um motor de jogos com código aberto. Porém, como o conceito é genérico, o escalonador pode ser implementado em outros motores de jogos.

O resto deste artigo está dividido da seguinte maneira: na seção 2 é apresentada brevemente o motor de jogos *Nightmare Fiction Framework.* A seção 3 apresenta o conceito de escalonador utilizado neste trabalho. As seções 4, 5 e 6 apresentam maiores detalhes da implementação do Escalonador de Cenas. A seção 4 é responsável por

apresentar o ambiente de desenvolvimento enquanto que as seções 5 e 6 apresentam, respectivamente, detalhes sobre o que consiste uma cena e o escalonador de cenas.

2. NF Framework

NF Framework é um motor de jogos totalmente aberto e que está em desenvolvimento utilizando a linguagem *C++* e *Modern OpenGL*. O motor ainda se encontra em estágio inicial de desenvolvimento, porém pretende ser portável para várias plataformas, inclusive consoles.

Figura 1. Fã game em desenvolvimento na NF Framework

A figura 1 ilustra o jogo que vem sendo desenvolvido utilizando o *Nightmare Fiction Framework*. Um fã game de **Resident Evil ©**, jogo originalmente desenvolvido pela **Capcom** para a plataforma **Playstation** no ano de 1998.

3. Escalonadores

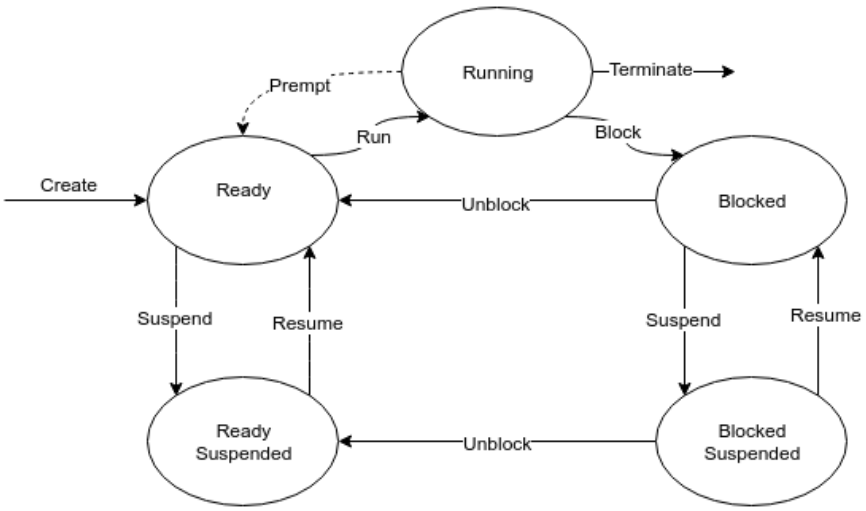

Figura 2. Funcionamento do escalonador
Fonte: Based on New York University - Operating Systems Diagram

Em Sistemas Operacionais, a chave para a multiprogramação é o escalonamento [Stallings, 2009]. O escalonador vai despachar o processo que deve ser executado no

momento. Como pode ser observado na Figura 2, cada processo tem um estado e dependendo do estado o escalonador vai realizar diferentes tarefas com ele (inicializar, executar, encerrar).

Outra forma de gerenciar atividades nos sistemas operacionais é com os escalonadores de *threads*, onde cada processo tem um conjunto de *threads* que vão executar independentemente e são gerenciadas pelo escalonador de *threads*.

4. Desenvolvimento

O escalonador foi implementado durante o desenvolvimento inicial da *NF Framework*, devido a necessidade de separar o código do motor do código do jogo. A vantagem de ter o código do jogo e motor separados é a maior organização e o desenvolvimento independente de partes dos códigos, facilitando sua manutenção. Além disto, o valor dessa separação se torna evidente quando os desenvolvedores começam a licenciar jogos e eles são adaptados com novas artes, cenários, armas, personagens, regras de jogo com apenas mudanças mínimas no código do motor de jogos [Jason, 2014].

O código do motor foi escrito em *C++* e a ideia é que o código do jogo só tenha acesso às referências dos *objetos* necessários para o seu desenvolvimento. Para resolver esse problema, foi utilizado o conceito de escalonadores de *threads* e *processos*. Em sistemas operacionais é comum existir um escalonador que vai despachar qual o processo que deve ser executado no momento. Qual processo ele vai executar e por quanto tempo, vai variar de acordo com o algoritmo implementado no escalonador.

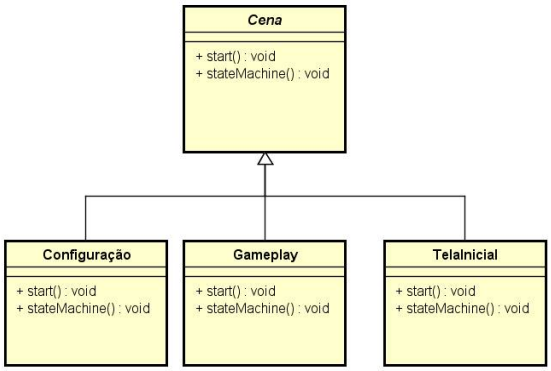

Figura 3. Funcionamento do escalonador

A partir dessa ideia de escalonador, temos o conceito de *Escalonador de Cenas*. Uma *cena* é uma classe genérica que vai utilizar o conceito de polimorfismo, onde as *classes* que herdam dela podem ter seu comportamento modificado através da herança. Como ilustrado na Figura 3, uma *cena* pode ser por exemplo, a *Tela Inicial* do jogo.

Para o gerenciamento das cenas, foi escrita uma outra *classe*, que é o

escalonador de cenas. O escalonador vai verificar qual a cena atual e vai passar para essa as referências dos objetos para essa classe, ele também deve verificar se a cena deve ser alterada, bem como verificar se é necessário limpar os recursos utilizados por ela.

Outro fator importante para o desenvolvimento do escalonador na *NF Framework* foram os *smart pointers*, recurso fornecido a partir do padrão *C++11*. Esse recurso de gerenciamento de memória dinâmica é muito importante pois cada cena é um objeto alocado dinamicamente. A vantagem de utilizar cenas alocadas dinamicamente é que: sempre que não haja a necessidade de utilizar a cena, o escalonador pode liberar a memória, mantendo assim um bom gerenciamento de memória.

5. Cena

A cena é uma classe genérica que vai ser herdada pelas classes principais do código do jogo. Ela possui uma máquina de estado, que é onde suas funcionalidades devem ser escritas. Ela também pode dizer ao escalonador qual a próxima cena a ser executada. Além disso, a cena também pode dizer ao escalonador através dos estados, o que deve fazer com a cena que está sendo executada.

5.1. Estados de uma cena

Os estados de uma cena servem para dizer ao escalonador de cenas o que deve ser feito com a cena que está sendo executada no momento. A figura 4 ilustra os estados de uma cena, abaixo a descrição de cada estado:

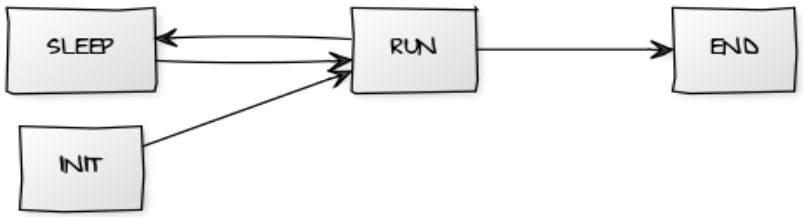

Figura 4. Estados possíveis em uma cena

- **INIT**: A cena deve ser inicializada, assim, deve-se passar todas as referências dos objetos para ela.
- **RUN**: A cena já foi inicializada, assim, chama-se a função que executa a máquina de estados da cena.
- **SLEEP**: O escalonador deve mudar para a próxima cena e o contexto da cena atual deve ser mantido.
- **END**: O escalonador deve mudar para a próxima cena e a cena atual deve ser liberada da memória assim como todos os recursos usados por ela.

5.2. Inicialização de uma cena

Essa função só vai ser executada apenas uma vez, quando o escalonador detecta que o estado atual é **INIT**, normalmente deve-se colocar aqui todos os recursos que vão ser carregados e inicializados pela cena, como carregamento de texturas, modelos, *shaders* [...].

5.3. **Máquina de estado de uma cena**

Toda vez que a cena estiver no estado **RUN**, o escalonador vai executar a função responsável pela máquina de estados. A máquina de estados deve ser onde o código da cena a ser executado deve se encontrar.

5.4. **Fim da execução de uma cena**

Se uma cena vai para o estado **END** ou **SLEEP** é necessário avisar ao escalonador qual a próxima cena a ser executada. No caso do **END** o escalonador deve liberar todos os recursos utilizados pela cena.

6. **Escalonador de Cenas**

O conceito de um escalonador de cenas é semelhante ao conceito de um escalonador de processos. O escalonador, que também pode ser chamado de **despachante**, vai ser executado com frequência e verificar o estado atual da cena que está sendo executada. As ações tomadas pelo *despachante* variam de acordo com o estado da cena.

O escalonador deve primeiro verificar qual o estado da cena atual, sua decisão é feita de acordo com o estado da cena que está sendo executada. As decisões que o escalonador pode tomar são as seguintes:

- Caso a cena esteja no estado **INIT**, o escalonador deve passar a referência dos objetos utilizado por essa cena e após isso chamar a função para inicializar os recursos da cena.
- Caso a cena esteja no estado **RUN**, o escalonador apenas vai executar a função de máquina de estados da cena.
- Caso a cena esteja no estado **SLEEP**, o escalonador deve escalonar a próxima cena e verificar se a cena está alocada. Caso não esteja, o escalonador deve alocar a próxima cena. Se a cena já estiver alocada, porém no estado **SLEEP**, o escalonador deve mudar o estado da cena para **RUN**.
- Caso a cena esteja no estado **END**, o escalonador deve pegar qual a próxima cena e verificar se a cena está alocada, caso não esteja, o escalonador deve alocar a próxima cena. Se a cena já estiver alocada, porém no estado **SLEEP**, o escalonador deve mudar o estado da cena para **RUN**. Como o estado da cena atual é **END**, O escalonador deve liberar todos os recursos alocados por ela.

O escalonador deve garantir que quando uma cena for encerrada os recursos alocados por ela devem ser liberados da memória. Além disso, também deve verificar se a próxima cena solicitada está pronta, caso contrário ele deve alocar a cena e inicializar. Se a cena está pronta, ele sempre executará a função da máquina de estados da cena. O despachante é inicializado pelo núcleo do motor de jogos. Ele deve passar informações como as referências dos objetos que o *despachante* deve providenciar para as cenas.

1. **Conclusões e Trabalhos futuros**

Neste artigo, a técnica *Escalonador de Cenas* foi apresentado como uma possível solução de separar o código do motor de jogos do código do jogo. O destaque dessa

solução é sua fácil implementação e eficiência ao garantir que recursos não utilizados sempre vão ser liberados.

Como trabalhos futuros, existem melhorias que podem ser acrescentadas na técnica de *Escalonadores de Cenas* como, por exemplo, o compartilhamento de recursos entre cenas. Atualmente cada cena tem seus próprios recursos e não há nenhum compartilhamento de informação entre elas. O próximo passo seria uma possível forma de enviar um *buffer* de informações entre cenas contendo as informações que devem ser compartilhadas. Outra melhoria é o paralelismo de cenas. O escalonador pode utilizar *threads* e executar duas cenas ao mesmo tempo. Isso seria útil quanto, por exemplo, precisamos de uma cena secundária que vai carregar os recursos (texturas, modelos 3d, cenários) enquanto a primária renderiza o que já foi carregado pela cena secundária.

Referências

[Stallings, 2009] Stallings, W (2009). Arquitetura e Organização de Computadores. 8. ed. São Paulo, Pearson.

[Jason, 2014] Jason Gregory (2014), Game Engine Architecture. 2. ed. CRC Press.

Allan Gottlieb - Operating Systems (2010-11 Spring), New York University. http://cs.nyu.edu/courses/spring11/G22.2250-001/lectures/lecture-04.html