Escola de Artes, Ciências e Humanidades - Universidade de São Paulo

Design Patterns e API Colletions aplicados ao jogo Shoot ‘em up

São Paulo

2014

Escola de Artes, Ciências e Humanidades - Universidade de São Paulo

Design Patterns e API Colletions aplicados ao jogo Shoot ‘em up

Computação Orientada a Objetos Turma 04

Adriano Augusto Satti Ortega Boschi - 8516142

Caio Vinicius Marques Teixeira - 8516883

Gerson Revoredo Pereira - 8598728

João Pedro Nardari dos Santos - 8623865

São Paulo

2014

Sumário

1. Críticas em relação ao código inicial

O código inicialmente apresentado no exercício programa apesar de utilizar a GameLib como biblioteca para implementar o jogo Shoot ‘em up. Apresenta um código totalmente estruturado, com muita repetição de código, não criação de herança para facilitar a composição dos objetos (enemies, player e background), uso excessivo de arrays e por sua extensão longa uma difícil manutenção e diversos pontos onde mudar qualquer parte do jogo necessita mexer em diversos pontos do código permitindo assim a geração de novos bugs ou parar com o funcionamento do game.

2. Nova estrutura de classes e Design Patterns utilizados

Para melhorar a implementação do código e seguir a proposta do exercício programa decidimos pela criação das seguintes classes: Background, Enemy, Enemy1, Enemy2, Enemy3, EP(classe principal), GameActor, GameController, GameEntity, GameLib, Player, PlayerProjectile, Projectile, Time e Vector.
Abaixo a descrição de cada uma e quais padrões de projeto foram adotados para a nova estrutura de classes.

Background

Define o comportamento do fundo do jogo, é filha de GameEntity.

Enemy

Classe abstrata que define os elementos comuns dos inimigos, como velocidade, rotação, etc. É filha de GameActor.

Enemy1

Define o comportamento do Enemy1, é filha de Enemy.

Enemy2

Define o comportamento do Enemy2, é filha de Enemy.

Enemy3

Define o comportamento do Enemy3, é filha de Enemy.

EP

Classe principal do exercício programa, antiga Main.java, deve-se compilar esta classe para executar o game.

GameActor

Filha de GameEntity, deve ser usada nos objetos que implementam colisões. Define o método:

- OnCollide(GameActor collider): Executado quando uma entidade colide com outra, passando a sua referência. Deve implementar a reação da entidade quando ocorrer a colisão, como explodir ao ser atingido por um projétil.

GameController

É responsável por controlar todo o jogo, implementa os padrões UpdateMethod (uma variação do padrão Observer) e o padrão Singleton.

O padrão UpdateMethod, consiste em manter listas com todas as entidades de jogo e iterar por todos os objetos executando seus métodos previamente definidos para atualizar seus estados (neste caso, update(), draw()).

A classe GameController contém duas listas:

- entities: Guarda todas as entidades de jogo.
- gameActors: Guarda todos os Actors do jogo.

O Loop é dividido nas seguintes etapas:

1. **Atualizar listas**: Verificar se foram adicionados ou deletados entidades na iteração anterior e atualizar as listas.
2. **Loop de física**: Iterar por todos os GameActors e verificar se há colisão entre cada par, se houver, chamar os respectivos métodos OnCollide() para tratar suas colisões.
3. **Loop update()**: Itera todas as entidades para executar seus métodos update() e atualizar seus estados.
4. **Atualizar inimigos**: Verifica se está na hora de criar novos inimigos
5. **Loop draw()**: Itera todas as entidades e executa seus métodos draw() para desenhar na tela.

GameEntity

Define um entidade de jogo, define os métodos:

- Update(): atualiza o estado da entidade
- Draw(): faz todas as operações de desenho da entidade

GameLib

Classe que já estava no projeto e não foi alterada, mini biblioteca com recursos utilizados pelas outras classes criadas na nova estrutura.

Player

Define o comportamento da nave do jogador, verifica e executa os controles. Filha de GameActor. Implementa o padrão de projetos Singleton para garantir que exista um único player no jogo.

PlayerProjectile

Define o comportamento dos projéteis atirados pelo jogador, é filha de Projectile pois sobreescreve seu método Draw().

Projectile

Define o comportamento dos projéteis atirados tanto pelos inimigos, é filha de GameActor.

Time

Responsável pelas operações baseadas em tempo, possui dois métodos:

- getCurrentTime(): Retorna o tempo decorrido após a execução do jogo.
- deltaTime(): Retorna o tempo corrido desde o último frame, é muito usado para operações que precisam de sincronização, como movimentação.

Vector

Define um vetor de duas variáveis (X e Y), é usado internamente para armazenar as posições das entidades. Também possui um método Distance() que retorna a distância entre dois pontos (definidos por objetos Vector).

3. Utilização da API Collections na nova estrutura adotada

São utilizados dois tipos de coleções no jogo: LinkedLists e Stacks. As listas são usadas para armazenar todas as entidades de jogo e as Stacks para guardar temporariamente os objetos instanciados em um frame para serem adicionados no próximo.

4. Conclusão

Podemos constatar que após a aplicação e implementação de uma nova estrutura de classes e metódos seguindo determinados padrões de projetos vistos em aula e outros obtidos através de pesquisa a manutenção tornou-se mais fácil no código que antes utilizava a programação estrutura para execução do jogo. Utilizando os design patterns voltados para jogos, conseguimos relacionar ainda mais os padrões citados na disciplina e observar a variação dos utilizados em jogos e os já conhecidos no livro Use a cabeça (Observer relacionado ao UpdateMethod). Além da aplicação de patterns, a utilização da API Collections facilitou a LinkedList o armazenamento das entidades pertencentes ao game e com Stack os objetos instanciados durante a execução facilitando a inserção e

remoção. Juntando todas as alterações feitas nesta nova estrutura, notamos uma melhora em relação ao projeto inicial, fornecendo uma fácil criação de novos objetos pertencentes ao jogo e manutenção do código já existente o que não era uma tarefa fácil no código inicial mantido na classe Main.java.

5. Referências Bibliográficas

FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES, B. Use a cabeça - Padrões de Projeto. 2ª ed. Brasil. Delta Books, 2007. 496.

Game Programming Patterns - Update Method. http://gameprogrammingpatterns.com/update-method.html. (Acessado em 29/07/2014).

RABIN, S. Introdução ao desenvolvimento de games - Vol 1. 1ª ed. Brasil. Cengage Learning, 2011. 162.