Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática
Ciência da Computação – 6° Semestre
Computação Distribuída
Profº Ismar Frango

Leila Akina Ino	10402951
Livia Alabarse dos Santos	10403046
Pedro Henrique Araujo Farias	10265432

REFATORAÇÃO DO *ASTEROIDS*

Sobre o código original

O código original disponibilizado claramente não segue os princípios SOLID. É fácil de perceber este fato na medida em que o código se apresenta como um enorme bloco de funções sem diferenciação de responsabilidades, denotando o descumprimento do primeiro princípio: o **princípio da responsabilidade única**.

Dessa forma, o código se torna mais difícil de se compreender e de se realizar manutenção. Portanto, no processo de **refatoração do código em microsserviços locais**, propõe-se que o código seja refatorado de modo que se aproxime mais dos princípios e boas práticas definidos pelo SOLID, modularizando-o e, assim, facilitando sua manutenção e seu entendimento.

Após refatoração

No processo de refatoração do *asteroids*, o código foi modularizado em algumas classes:

- ***Object***: Classe mãe das classes *Player*, *Bullet* e *Asteroid*. Nela, está definido o método construtor, reutilizado pelas classes filhas;
- ***Player***: Classe responsável pelas funcionalidades relacionadas à nave controlada pelo jogador. Classe filha de *Object*. Implementa métodos de *draw* e *move*, abstraídas pelas classes *Draw* e *Movement*;
- ***Bullet***: Classe responsável pelas funcionalidades relacionadas às balas disparadas. Classe filha de *Object.* Implementa métodos de *draw* e *move*, abstraídas pelas classes *Draw* e *Movement*;

- ***Asteroid***: Classe responsável pelas funcionalidades relacionadas aos asteroides. Classe filha de *Object*. Implementa métodos de *draw* e *move*, abstraídas pelas classes *Draw* e *Movement*;
- ***Draw***: Classe responsável por abstrair o método de desenho de uma instância de *Object*;
- ***Movement***: Classe responsável por abstrair o método de movimento de uma instância de *Object*;
- ***Collision***: Classe responsável pela verificação de colisão entre asteroide e bala e entre asteroide e jogador;
- ***MainGame***: Classe responsável pela execução, segundo a lógica estabelecida para o jogo. Utiliza as classes *Draw* e *Movement* para executar os métodos abstraídos por estas em diferentes objetos;
- ***Main***: Classe responsável por iniciar execução do programa;

É possível perceber que, a partir da modularização do programa nas classes citadas, as quais apresentam responsabilidades específicas e bem definidas, buscou-se satisfazer o princípio da responsabilidade única. Isso fica explícito ao perceber a criação de uma classe *Player* para realizar operações com a nave controlada pelo jogador e de uma classe *Asteroid* para realizar operações com os asteroides, por exemplo.

Além disso, as classes *Draw* e *Movement* têm o objetivo de abstrair os métodos *draw* e *move* de outras classes, como *Asteroid*, *Bullet* e *Player*. Sendo assim, essas duas classes representam espécies de interfaces, sendo implementadas somente por classes que utilizam os métodos abstraídos nestas. Dessa forma, nota-se o princípio de segregação de interface, o qual afirma que uma classe não deve ser forçada a implementar interfaces que não utilizará.

Obs.: Como não identificamos *interfaces* em Python, *Draw* e *Movement* foram implementados como classes, embora sua intenção seja a de uma *interface*.

Por fim, vale citar também o princípio *Open-Closed*, o qual afirma que uma classe deve estar aberta para extensão, mas fechada para modificação. Isso é observado nas classes *Object*, *Asteroid*, *Bullet* e *Player*. Ao invés de adicionar uma série de *if*s e *else*s na classe *Object* para especificar sua natureza, cria-se outras três classes que extendem-na, herdando-a. Dessa forma, evitamos modificação na classe *Object* e adicionamos funcionalidades a suas herdeiras.

Diagrama UML de classes