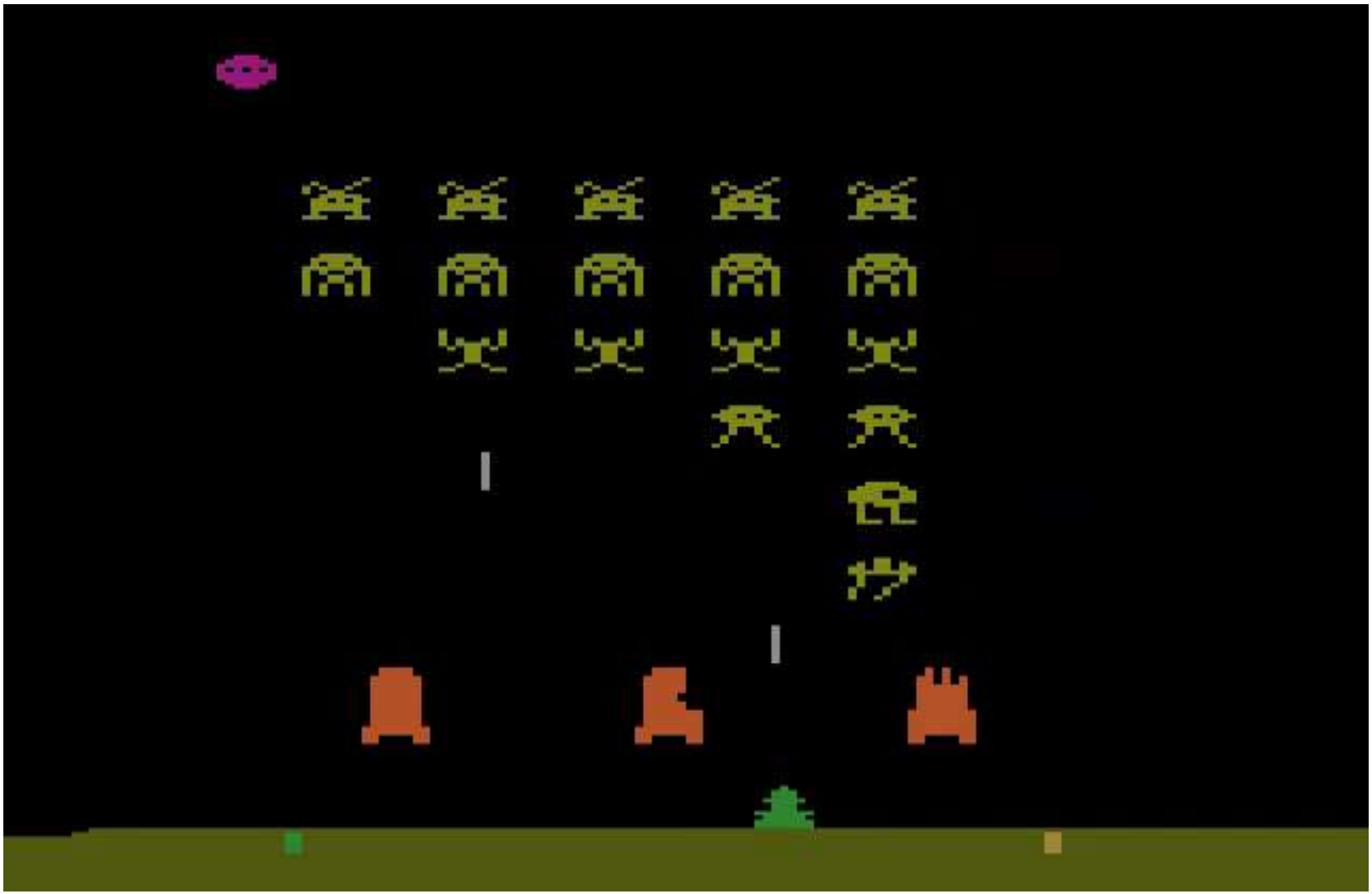
Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.2

http://www.dcc.ufrj.br/~fabiom/java

Space Invaders



Componentes do Jogo

- Canhão
- Aliens
- Tiros
- Escudos
- Score e vidas
- Chão
- Nem todos vão precisar de classes próprias para representa-los!

A classe `Hitbox`

- A caixa de colisão deve poder testar se ela colidiu com outra caixa de colisão (e em que lado dessa outra caixa):

```
public class Hitbox
{
    public static int TOPO     = 1;
    public static int ESQUERDO = 2;
    public static int FUNDO    = 4;
    public static int DIREITO  = 8;

    // Canto superior esquerdo e
    // inferior direito
    double x0, y0, x1, y1;
    ...
}
```

```
// Esse retângulo colidiu com hb, e onde em hb?
public int intersecao(Hitbox hb) {
    double w = ((x1-x0) + (hb.x1 - hb.x0)) / 2;
    double h = ((y1-y0) + (hb.y1 - hb.y0)) / 2;
    double dx = ((x1 + x0) - (hb.x1 + hb.x0)) / 2;
    double dy = ((y1 + y0) - (hb.y1 + hb.y0)) / 2;
    if (Math.abs(dx) <= w && Math.abs(dy) <= h) {
        double wy = w * dy; double hx = h * dx;
        if (wy > hx) {
            if (wy > -hx) return FUNDO;
            else return ESQUERDO;
        } else {
            if (wy > -hx) return DIREITO;
            else return TOPO;
        }
    }
    return 0;
}
```

Caixas de colisão no Space Invaders

- Cada alien tem uma caixa de colisão ligeiramente menor que ele
- O canhão tem uma caixa de colisão do tamanho da sua base
- Os tiros têm caixas de colisão do tamanho exato de cada um
- Podemos representar os escudos como uma coleção de pequenos tijolos, cada um com sua caixa de colisão

Coordenando as colisões

- Para saber se as colisões aconteceram o coordenador usa as caixas de colisão dos tiros e dos outros objetos
- Se alguma colisão aconteceu o coordenador toma a ação apropriada
- Caso o tiro acerte algum objeto ele é avisado para fazer a sua animação de destruição apropriada
- Em um primeiro momento a lógica do método `tique` do coordenador vai ficar muito grande: isso é um sinal de que devemos *refatorar* esse método em diferentes métodos que cuidam de cada parte da lógica de atualização do jogo

sem mudar funcionalidade

reescrever p/ a lógica mais transparente

Princípios de projeto OO

- Estamos procurando seguir dois princípios básicos do projeto de programas OO
- O primeiro diz que métodos de um objeto não devem modificar diretamente campos de outro objeto
- O segundo diz que métodos devem ser curtos e terem uma função bem clara
- Numa primeira implementação podemos violar esses princípios, mas depois sempre devemos voltar e procurar resolver essas violações criando novos métodos e delegando para eles

Composição

- *Composição* é a ferramenta principal da modelagem OO: objetos são compostos por outros objetos

- A composição anda de mãos dadas com a *delegação*: um objeto deve sempre delegar parte da implementação de suas operações para suas partes

  - Em geral, se estamos usando apenas os campos de um objeto, está faltando delegação na modelagem

- Estamos usando composição e delegação desde o início em nossos exemplos

Um timer para o Space Invaders

- Como mais um exemplo de composição e delegação, vamos adicionar um timer que os vários objetos do jogo poderão usar

- Quando criamos o timer dizemos quanto tempo queremos que ele conte, e depois vamos acumulando intervalos de tempo

- Podemos verificar também se o timer “disparou”, e resetar o timer

Visibilidade

- Nem todos os campos e operações de um objeto são para consumo externo; várias delas podem ser apenas para uso pelo próprio objeto

- Em Java, podemos marcar qual a *visibilidade* de um campo ou um método:
  - `public` indica que o acesso é livre
  - `private` indica que o acesso é restrito apenas às instâncias da classe

- Quando não dizemos nada, temos um campo ou método que é público para quem estiver na mesma pasta, e privado para o resto