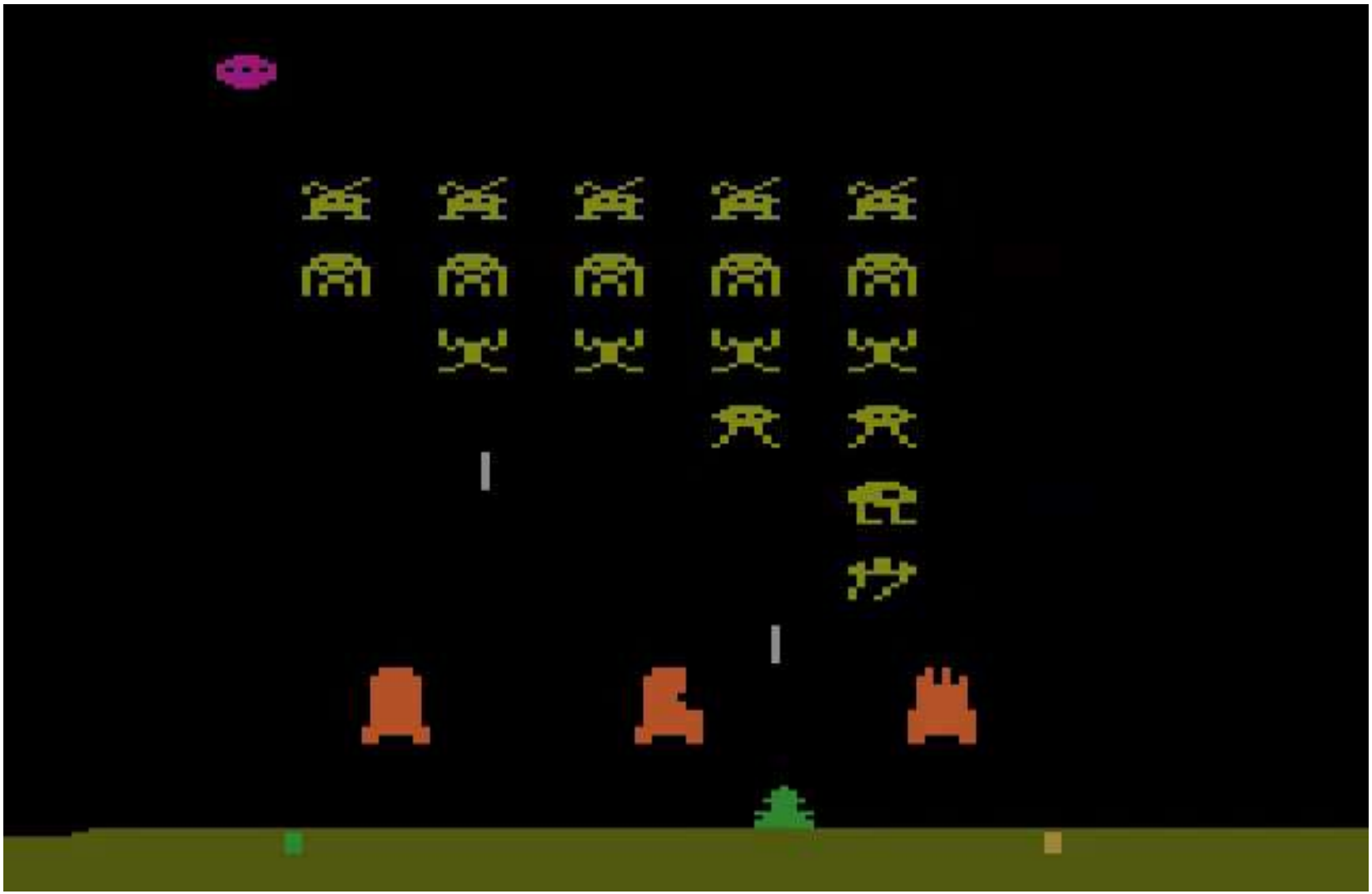
Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.1

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!

Interação com o motor de jogo

- O motor interage com nosso jogo mandando mensagens para o objeto principal do jogo, ou seja, chamando seus métodos

- O método `desenhar` avisa ao jogo que ele deve desenhar um quadro da sua interface; como parâmetro, o jogo recebe um objeto que representa a tela de desenho

- O método `tique` avisa ao jogo da passagem de tempo, para ele atualizar seu estado interno; como parâmetros, o jogo recebe quantos segundos se passaram, e quais teclas o jogador está pressionando no momento

- O método tecla avisa do jogo que uma tecla foi solta, informando qual foi essa tecla

O objeto `tela`

- O objeto tela responde a cinco métodos de desenho:

```
public void triangulo(double x1, double y1,
        double x2, double y2, double x3, double y3, Cor cor)
public void circulo(double cx, double cy, int raio, Cor cor)
public void quadrado(double x, double y, int lado, Cor cor)
public void retangulo(double x, double y, int largura, int altura, Cor cor)
public void texto(String texto, double x, double y, int tamanho, Cor cor)
public void imagem(String arquivo, int xa, int ya, int larg, int alt,
                   double dir, double x, double y)
```

- A posição para retângulos e imagens é a do canto superior esquerdo; para o círculo é a do centro, para triângulos dos vértices, e para o texto do canto inferior esquerdo

- Podemos criar uma cor com uma tripla de componentes vermelho, verde e azul inteiros, onde 0 é ausência e 255 a intensidade máxima, decimais, onde 0.0 é ausência e 1.0 a intensidade máxima, ou usar alguma das cores pré-definidas como variáveis globais de `Cor`

Desenhando o jogo

- Começamos pelo método `desenhar`
- Poderíamos desenhar tudo acessando os campos dos nossos objetos e chamando os métodos apropriados no objeto `tela`, mas isso não é um bom projeto
- Saber se desenhar deve ser responsabilidade de cada um dos objetos do jogo
- Fazemos isso definindo métodos `desenhar` em cada uma das classes do jogo: `Canhao`, `Alien` e `Tiro`, e *delegando* a tarefa de desenhar para as instâncias dessas classes

Interagindo com o usuário

- Vamos ter dois tipos de interação com o usuário: segurar a seta esquerda move o canhão para a esquerda, segurar a seta direita move o canhão para a direita, apertar a tecla de espaço dispara um tiro

- Verificamos o estado das setas respondendo ao método `tique`:

```
public void tique(HashSet<String> teclas, double dt) {
    if(teclas.contains("left")) c.esquerda(dt);
    if(teclas.contains("right")) c.direita(dt);
    ...
}
```

- Verificamos a tecla de espaço respondendo ao método `tecla`:

```
public void tecla(String t) {
    if(t.equals(" ")) tiro();
}
```

- Note que em ambos os casos delegamos o efeito no estado do jogo a outros métodos

Animando o canhão os aliens

- Podemos completar o método tique para fazer o movimento do canhão, dos aliens e dos tiros, novamente delegando isso para esses objetos:

```
c.esquerda(dt)
c.direita(dt)
for(Alien a: as) a.mover(dt);
for(Tiro t: ts) t.mover(dt);
```

- O código de cada movimento em si é simples, mas é um bom projeto sempre delegar para o objeto qualquer mudança em seu estado interno

Coordenação

- Tanto no Space Invaders como em outros jogos, precisamos verificar possíveis colisões entre os objetos do jogo, e tomar ações a depender de qual objeto colidiu com qual
  - Ex: *se o tiro colide com um alien, o alien é destruído e o tiro some*
- Verificar uma interação envolvendo campos de dois (ou mais) objetos diferentes, e disparar ações em todos eles, é um problema da modelagem OO
- De quem é a responsabilidade de *coordenar* essa interação?

Coordenação, cont.

- Podemos eleger um dos objetos que estão participando da interação para ser o coordenador, mas isso aumenta o *acoplamento* entre os objetos que estão interagindo

- Ou podemos usar um mediador, um objeto que vai verificar se houve alguma interação, e mandar os objetos envolvidos tomarem uma ação

- O mediador precisa ter acesso ao estado dos objetos que estão interagindo, mas o acoplamento entre esses objetos diminui

- Vamos usar a instância de `Jogo` como mediador em nosso exemplo

Testando colisões

- Testamos se dois elementos do jogo colidiram testando se eles têm alguma interseção

- Jogos costumam simplificar esse problema com uma convenção, assumindo que todos os elementos são retângulos, independente da forma real deles: esses retângulos são as *caixas de colisão* ou *hitbox*

- Um elemento pode até ter mais de uma caixa de colisão, representando diferentes áreas dele, e elas vão acompanhando ele à medida que ele se move pela tela

- Podemos modelar caixas de colisão com uma classe própria, e nessa classe implementar a lógica para testar a interseção entre duas caixas de colisão

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