Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.2

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

Comunicação framework vs. jogo

- Toda a comunicação do framework com a jogo se dá através de métodos

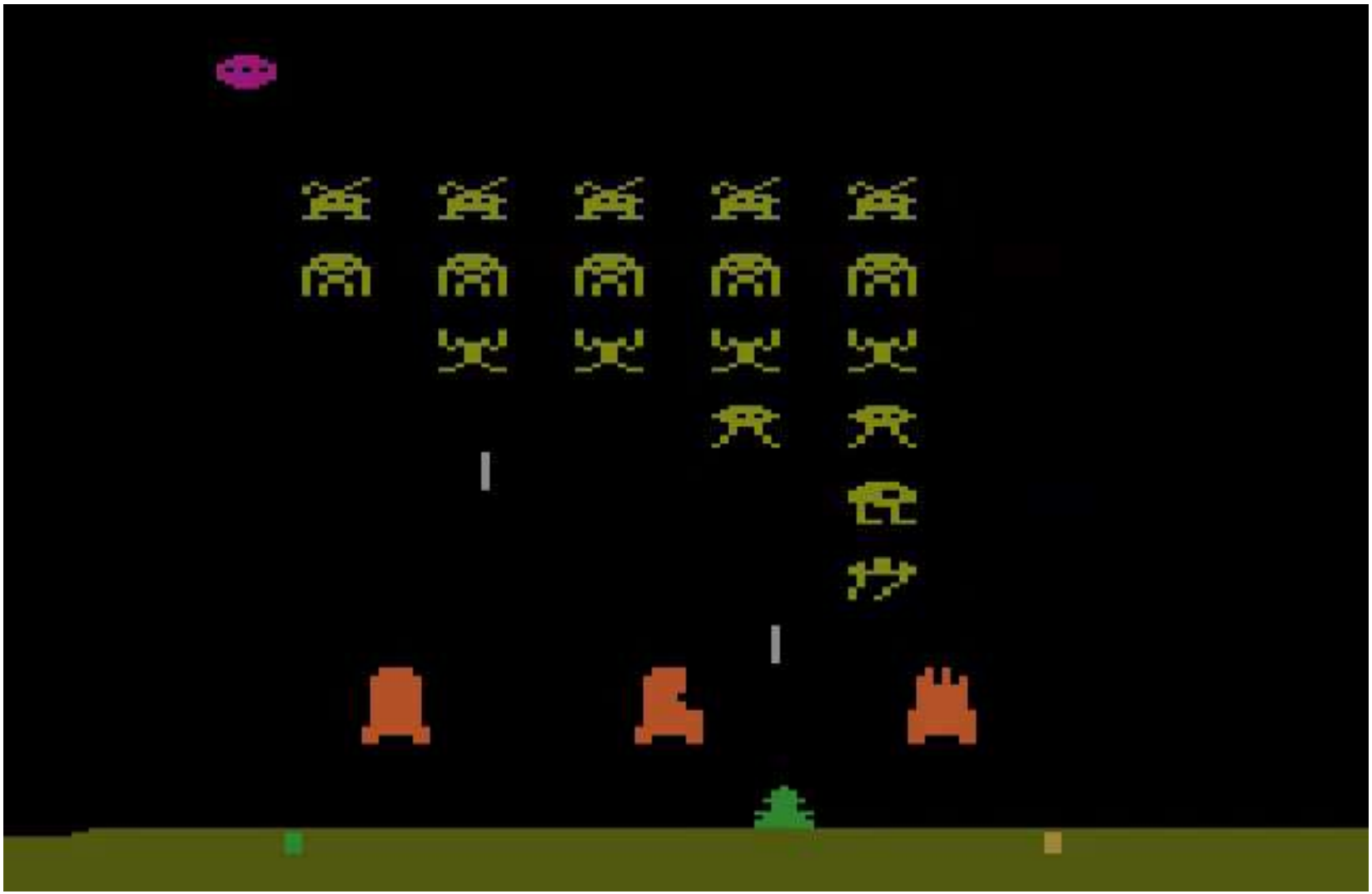
- O jogo (uma classe `Jogo`) define seis métodos

```
String getTitulo()
int getAltura()
int getLargura()
void tecla(String tecla)
void tique(Set<String> teclas, double dt)
void desenhar(Tela tela)
```

- A classe `Tela` define outros seis

```
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)
```

Space Invaders



Componentes do Jogo

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

Canhão

- Representamos o canhão com uma posição e uma velocidade, e com informações para desenhar o canhão

- A posição tem um componente horizontal e um vertical

```
public class Canhao {
    double x;
    double y;
    double vx;
    ...
}
```

- Vários dos campos têm valores fixos, então vamos ver uma forma melhor de inicializá-los

Inicialização fora do construtor

- Às vezes queremos que todo campo de um objeto seja inicializado da mesma forma, independente do que foi passado para o construtor

- Podemos fazer a inicialização direto na declaração do campo:

  - `int vx = 300;`

- Todo objeto criado vai começar com `vx` igual a `300`, mas depois o valor de `vx` de cada um pode divergir

Inicialização fora do construtor, cont.

- A inicialização é feita sempre que um novo objeto for criado, então cada objeto pode receber um valor diferente:
  - `public Cor cor = new Cor(Math.random(), Math.random(), Math.random());`
- Cada objeto criado vai receber **uma nova cor** em que cada componente é determinado aleatoriamente
- É como se a inicialização estivesse sendo feita dentro de **cada** construtor da classe

Campos da classe (*campos estáticos*)

- E se queremos que **todos** os objetos de determinada classe tenham um campo que sempre vai ter o **mesmo** valor?

- Nesse caso, usamos um *campo estático*, declarado com a palavra-chave `static`

  - `static int tamanho = 30;`

- Podemos acessar campos da classe por qualquer instância, ou diretamente pela classe

  - `Alien.largura, canhao.vx`

Campos da classe, cont.

- Para a linguagem, não há diferença entre campos estáticos e variáveis globais; a diferença está no que elas representam
- Uma variável global está associada sintaticamente a uma classe, mas é um valor que não está ligado ao comportamento de suas instâncias
  - Exemplos de cores pré-alocadas dentro da classe `Cor`
- Um campo da classe é um valor que afeta o comportamento de todas as instâncias da classe, e uma maneira de fazer elas compartilharem algum dado
  - Tamanho dos aliens e velocidade do canhão no *space invaders*

Aliens

- Cada alien tem uma posição e uma velocidade

```
public class Alien {
    double x;
    double y;
    double vx;
    ...
}
```

- Todos os aliens têm sempre o mesmo tamanho, e podemos usar variáveis globais

- Podemos armazenar os aliens em um vetor dentro na classe `Jogo`

Tiros

- Um tiro também tem uma posição e uma velocidade, mas a velocidade agora é vertical:

```
public class Tiro {
    double x;
    double y;
    double vy;
    ...
}
```

- Estamos criando novos tiros a todo momento, então um vetor não é a estrutura mais adequada

- Vamos usar uma das *coleções* de Java

Classes parametrizadas (*classes genéricas*)

- Várias classes da biblioteca padrão de Java têm *parâmetros*
- Por exemplo, todas as classes que representam coleções (listas, mapas, conjuntos) recebem um parâmetro que diz quais são os objetos que fazem parte da coleção
- Os parâmetros são outras classes, e aparecem entre `<>`
  - `HashSet<String>`, `ArrayList<Tiro>`, `HashMap<String,Aluno>`
- Depois veremos como definir nossas próprias classes parametrizadas

As classe `HashSet` e `ArrayList`

- Duas das coleções mais comuns usadas em programas Java são as classes `HashSet` e `ArrayList`

- Respectivamente elas representam conjuntos e listas de elementos, e são parametrizadas pelo tipo dos elementos que você quer guardar

- Use um conjunto se você não se importa com a ordem dos elementos, mas quer checar se um objeto pertence ao conjunto ou não, e não quer ter elementos duplicados

- Use uma lista se a ordem em que os elementos está importa, e você quer acessar um elemento pela posição dele na lista

Laço `for` de coleções

- Java tem uma versão do laço for que é especializada para percorrer uma coleção (um vetor, ou instâncias de uma das classes de coleção como `HashSet` e `ArrayList`)

- O bloco do laço é executado para cada elemento da coleção, com a variável de controle apontando para o elemento

```
for(int i: vetor) {
    System.out.println(i);
}
```

Chão e Score

- O chão é fixo, enquanto o score é só um valor escalar, então podemos manter os dados necessários para ambos na própria classe que representa o estado do jogo

- Nessa classe também instanciamos os objetos iniciais: o canhão e os aliens

```
public class Jogo {
    Canhao c;
    Alien[] as;
    ArrayList<Tiro> ts;
    int score;
    int vidas;
    boolean gameOver;
    ...
}
```

```
public Jogo() {
    c = new Canhao(getLargura()/2, getAltura() - 50, 300);
    as = new Alien[5 * 6];
    for(int i = 0; i < as.length; i++ ) {
        int lin = i / 5;
        int col = i % 5;
        as[i] = new Alien(col * Alien.larg + Alien.larg,
                          lin * Alien.alt + 200);
    }
    vidas = 3;
}
```