

# INF216

2023/2



# Projeto e Implementação de Jogos Digitais

## A2: Game Loop e Modelagem de Objetos

# Logística

## Avisos

- ▶ Entrega: P1 – Configuração Inicial: sexta-feira, às 07:30h!
- ▶ Aceitar o convite para o Slack!

## Última aula

- ▶ Organização da disciplina

# Plano de Aula

- ▶ Game Loop
  - ▶ Eventos de entrada
  - ▶ Atualização de objetos do jogo
  - ▶ Geração de saída
  - ▶ Gerenciamento do tempo do jogo
- ▶ Modelagem de Objetos
  - ▶ Modelo de hierarquia de classes
  - ▶ Modelo de componentes
  - ▶ Modelo híbrido

# Game Loop

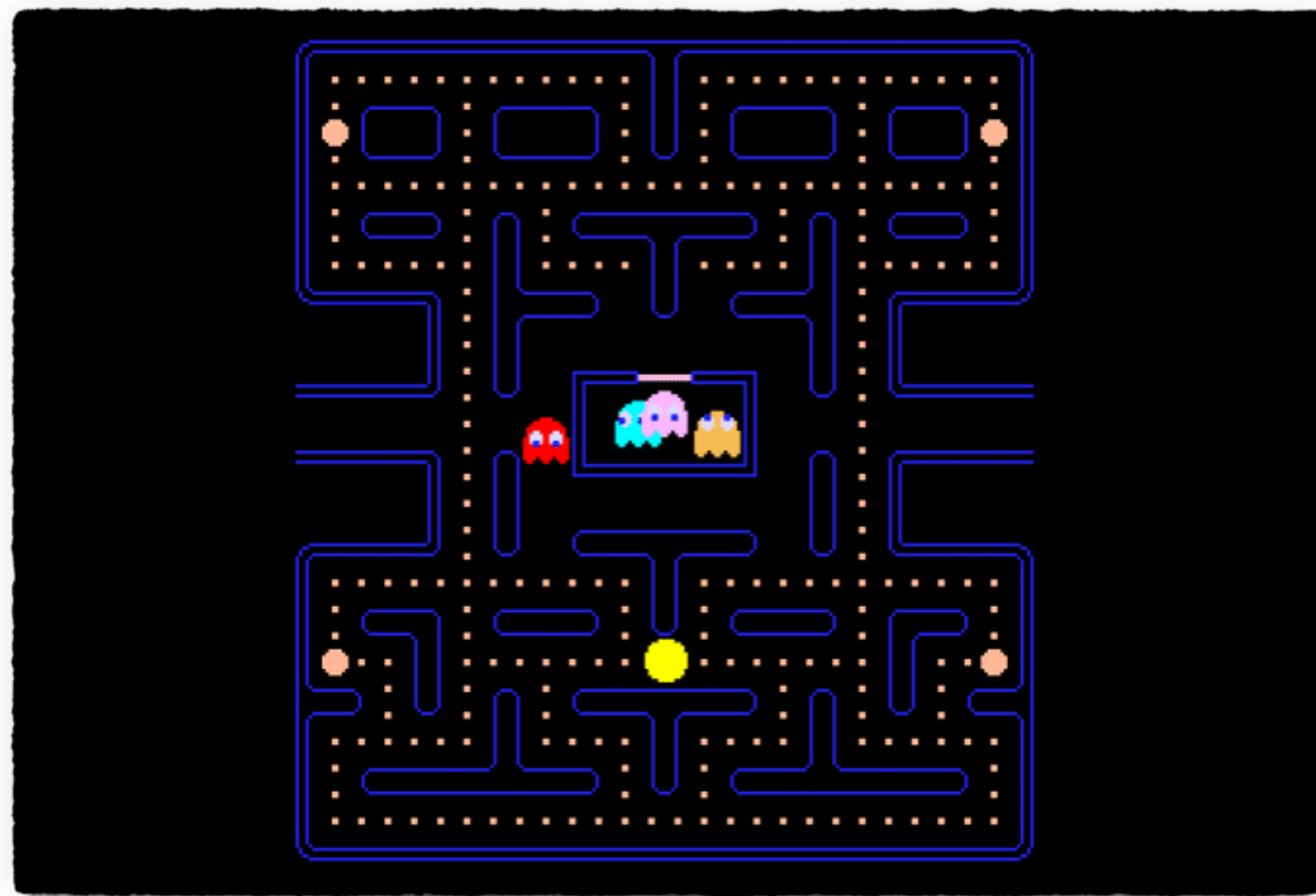
# Execução em lote (batch)

Em geral, os programas que escrevemos no curso de computação são executados em lote:

```
1. int x;  
2. cin >> x;  
3. cout << 2 * x << endl;
```

# Game Loop

Jogos digitais são executados em tempo real.

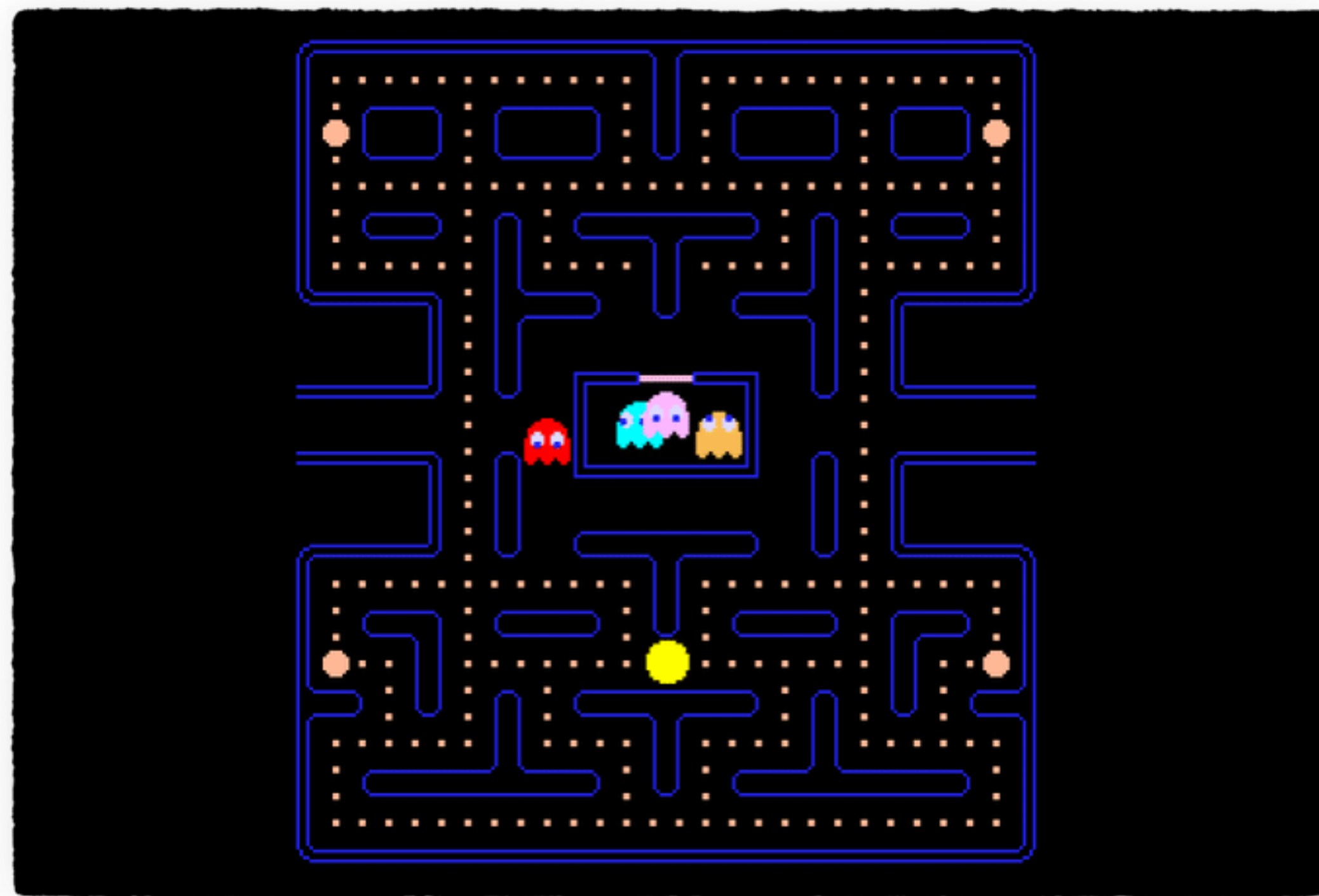


*Pacman - Namco, 1980 (arcade)*

```
while Game is running
    Process input
    Update game world
    Generate output
```

# Game Loop

Leitura de eventos de entrada.



*Pacman - Namco, 1980 (arcade)*

**while** Game is running

▶ Process input

▶  Keyboard

▶  Mouse

▶  Joystick

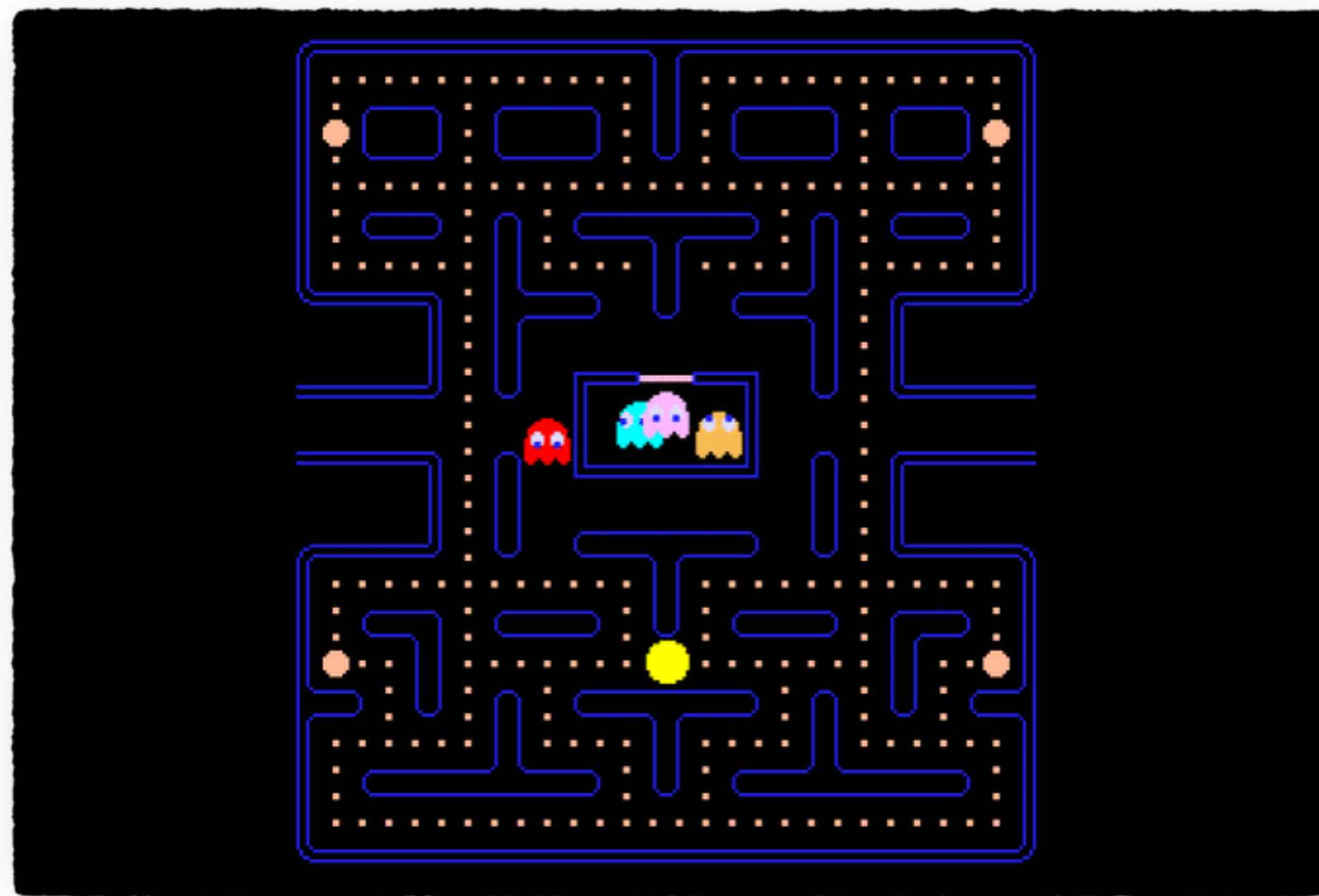
▶ ...?

Update game world

Generate output

# Game Loop

Atualização do estado de todos os objetos do mundo do jogo.



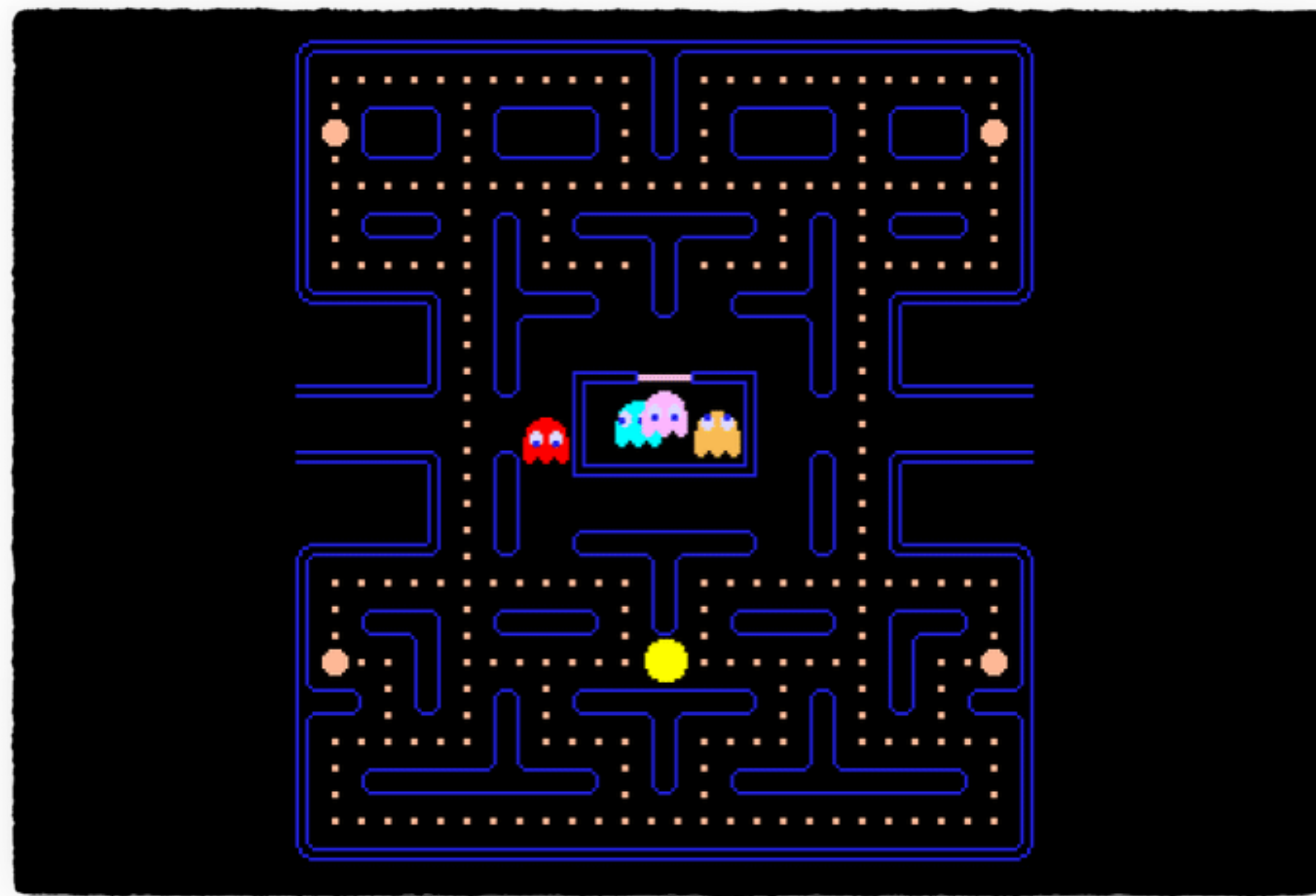
*Pacman - Namco, 1980 (arcade)*

```
while Game is running
    Process input
    ▶ Update game world
        ▶ Pacman
        ▶ Ghosts
        ▶ ...?
    Generate output
```



# Game Loop

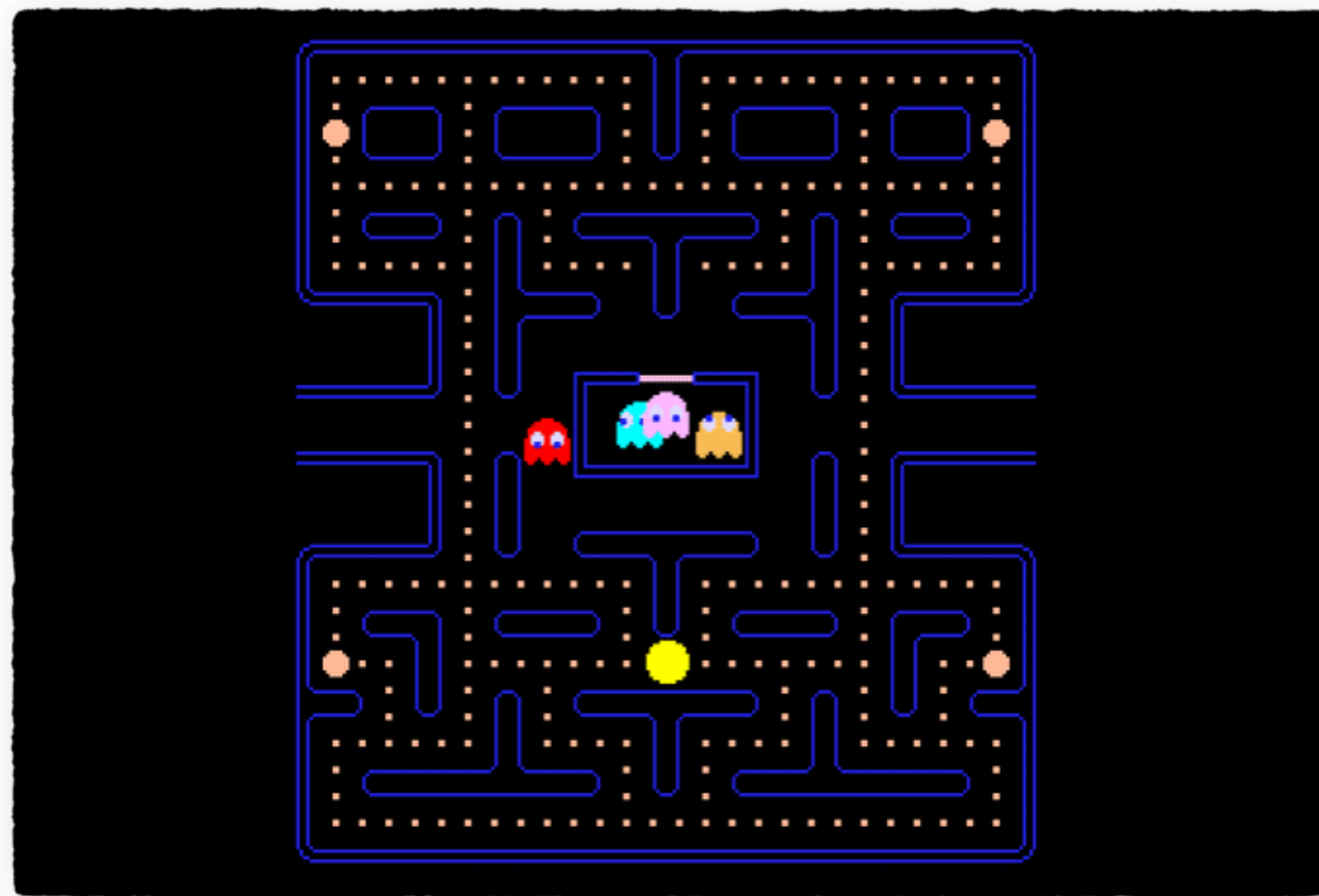
Gerar saídas com base no estado atualizado dos objetos do mundo.



*Pacman - Namco, 1980 (arcade)*

```
while Game is running
    Process input
    Update game world
    ▶ Generate output (frame)
        ▶ Image
        ▶ Sound
        ▶ ...?
```

# Pseudocódigo do PacMan



*Pacman - Namco, 1980 (arcade)*

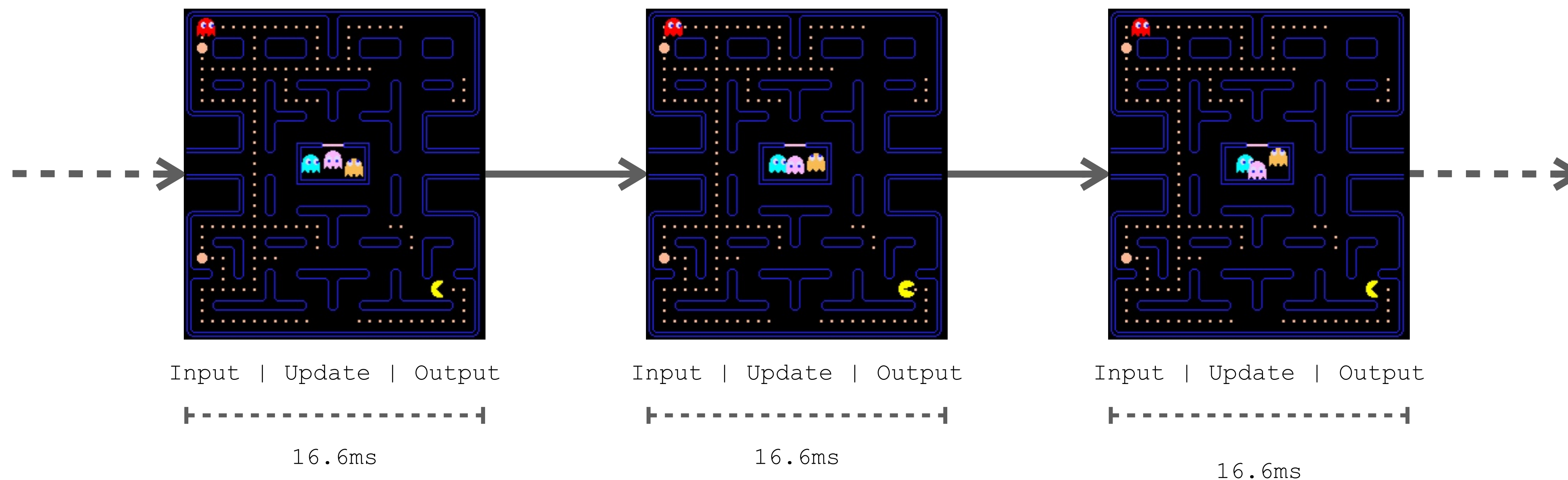
```
while player.lives > 0
    // Processar entrada
    input = read raw input data

    // Atualizar o mundo do jogo
    update player.position based on input
    foreach Ghost g in world
        if player collides with g
            kill either player or g
        else
            update AI for g

    // Comer as pastilhas
    // Gerar saídas
    draw graphics
    play audio
```

# Execução em lote vs tempo-real

Pense em um jogo como uma sequência de quadros, cada um com um limite de tempo definido.



Por exemplo: 60 FPS,  $1/60 = 16.6\text{ms}$  por quadro

# Tempo do jogo

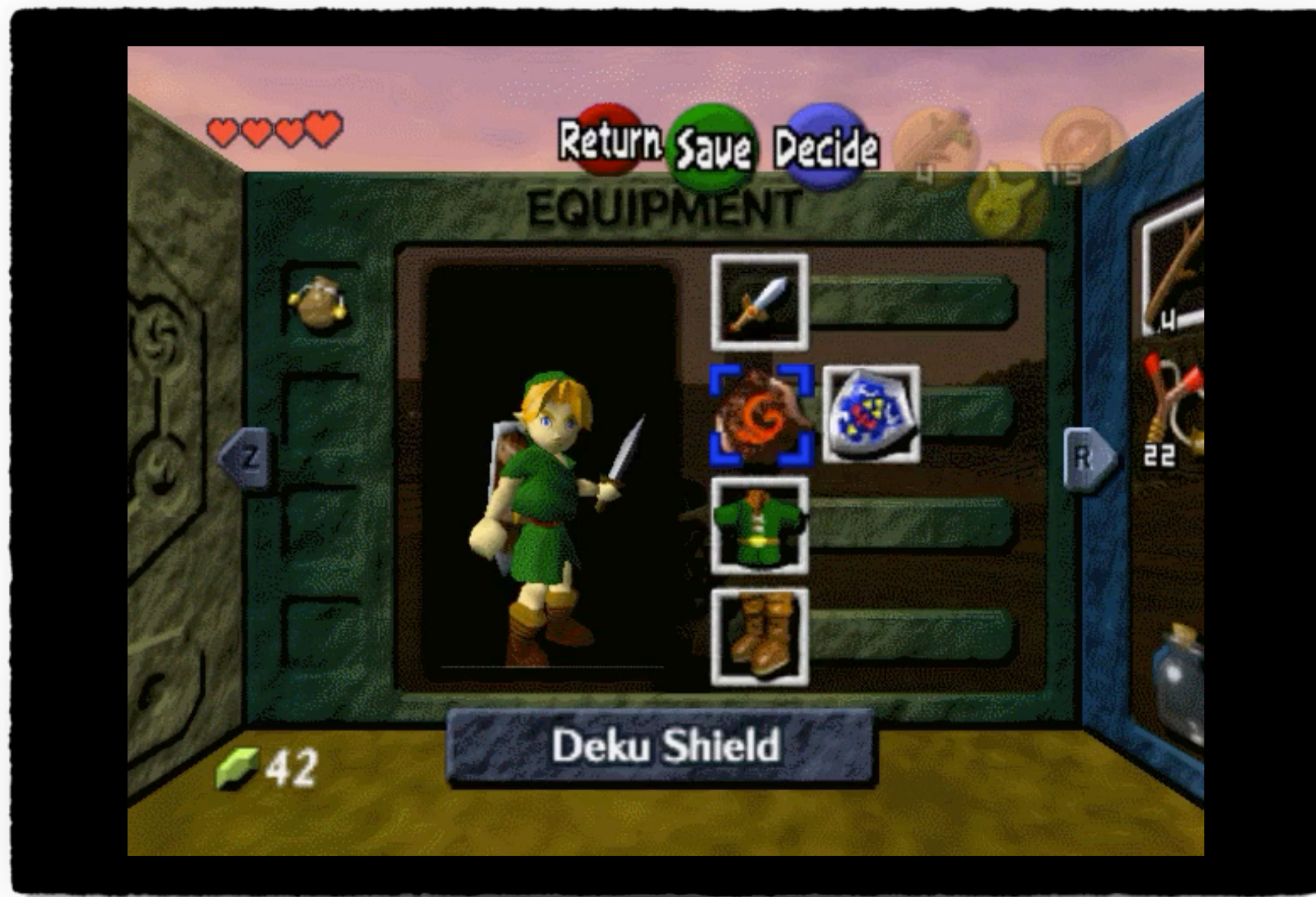
O tempo do jogo é diferente do tempo de relógio:

- ▶ Parar o tempo (game pause)
- ▶ Mais rápido
- ▶ Mais devagar
- ▶ Voltar no tempo



# Parar o tempo

Em quase todos os jogos modernos, o jogador pode pausar e resumir o jogo quando quiser.





# Mais rápido

Em alguns jogos, como os de esporte, o tempo costuma passar mais rápido, para que a partida seja mais rápida do que na vida real.



# Mais devagar



Em outros, o tempo passa mais devagar para que o jogador tenha mais tempo para planejar e executar uma ação.



# Voltar no tempo



Além disso, em alguns jogos, o jogador pode voltar no tempo como parte da própria mecânica do jogo.



# Game Loop

```
while (gameIsRunning) {  
    ProcessInput();  
    Update();  
    GenerateOutput();  
}
```

**Qual o problema com esse loop?**

Nenhum controle sobre o tempo!

# Gerenciamento do tempo do jogo

Existem três técnicas principais para gerenciamento do tempo do jogo

- ▶ Intervalos de tempo (FPS) fixos
- ▶ Intervalos de tempo (FPS) variáveis
- ▶ Intervalos de tempo Fixos (FPS) apenas para o Update

# Game Loop

## Intervalos de tempo (FPS) fixos

```
MS_PER_FRAME = 16.6
```

```
while (gameIsRunning) {  
    start = getCurrentTime();  
    ProcessInput();  
    Update();  
    GenerateOutput();  
    Sleep(start + MS_PER_FRAME - getCurrentTime())  
}
```

**Dormir no tempo que sobrou!**

## Qual o problema com esse loop?

Se o jogo rodar muito devagar, o tempo de dormir fica negativo.

# Game Loop

## Intervalos de tempo (FPS) variáveis

```
lastTime = getCurrentTime();  
while (gameIsRunning) {  
    deltaTime = (getCurrentTime() - lastTime) / 1000;  
    ProcessInput();  
    Update(deltaTime); Todos os objetos são atualizados em função de deltaTime!  
    GenerateOutput();  
    lastTime = getCurrentTime();  
}  
  
// Atualiza a posição x por 150 pixels/segundo  
position.x += 150 * deltaTime;
```

# Game Loop

## Intervalos de tempo (FPS) variáveis

```
// Atualiza a posição x por 150 pixels/segundo  
position.x += 150 * deltaTime;
```

**Quantos pixels esse objeto se move em 1 segundo se o jogo é atualizado a:**

**(a) 30 quadros por segundo?**

A 30 FPS, o delta time é  $\sim 0.033$ , então o objeto irá se mover a 5 pixels/quadro;  
 $30 * 5 = 150$  pixels/segundo.

**(b) 60 quadros por segundo?**

A 60 FPS, o delta time é  $\sim 0.016$ , então o objeto irá se mover a 2.5 pixels por quadro;  
 $60 * 2.5 = 150$  pixels/segundo.

**Mesma quantidade de movimento, mas com 60 FPS será mais suave!**

# Game Loop

Intervalos de tempo (FPS) variáveis

## Qual o problema com delta time loop?

1. Física e IA instáveis (e.g., pulos diferentes em jogos de plataforma)
2. Delta time muito grande (e.g., durante debug)

# Game Loop

## Intervalo de tempo (FPS) fixo apenas para o Update

```
MS_PER_FRAME = 16.6
```

```
MAX_DELTA_TIME = 0.05
```

```
lastTime = getCurrentTime();
```

```
while (gameIsRunning) {
```

```
    ProcessInput();
```

```
    Sleep(lastTime + MS_PER_FRAME - getCurrentTime())
```

```
    |-----|
```

```
    deltaTime = (getCurrentTime() - lastTime) / 1000;
```

```
    if deltaTime > MAX_DELTA_TIME:
```

```
        deltaTime = MAX_DELTA_TIME
```

```
    lastTime = getCurrentTime();
```

```
    Update(deltaTime);
```

```
    GenerateOutput();
```

```
}
```

**1. Esperar o tempo que falta para completar 16.6ms desde o último Update!**

**2. Limitar deltaTime!**

# Modelagem de Objetos



# Objetos do jogo

Objetos de jogos têm comportamentos diferentes!



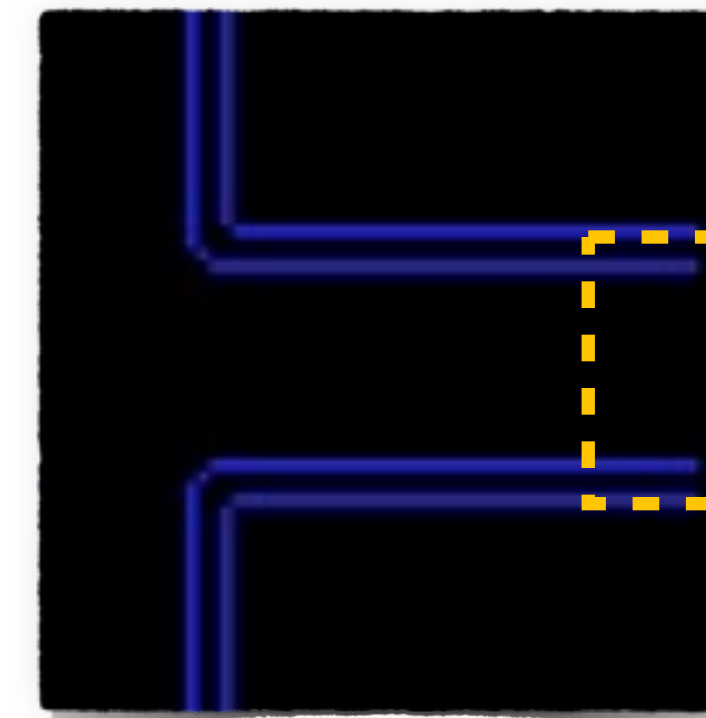
## Objetos Dinâmicos

- ▶ Draw
- ▶ Update
- ▶ Animate



## Objetos Estáticos

- ▶ Draw



## Gatilhos (triggers)

- ▶ Update

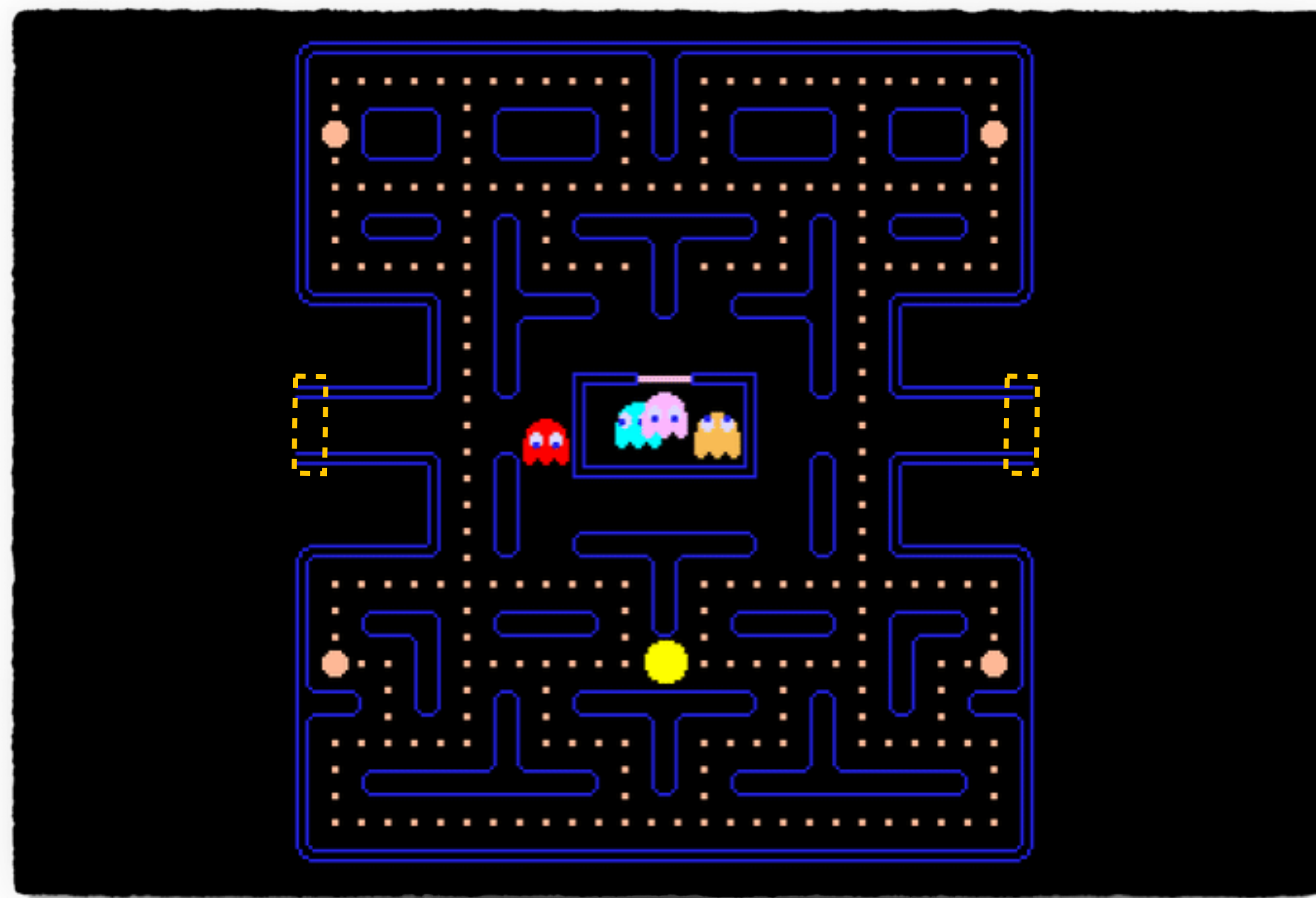
# Modelagem de Objetos

Existem três técnicas principais para modelagem de objetos:

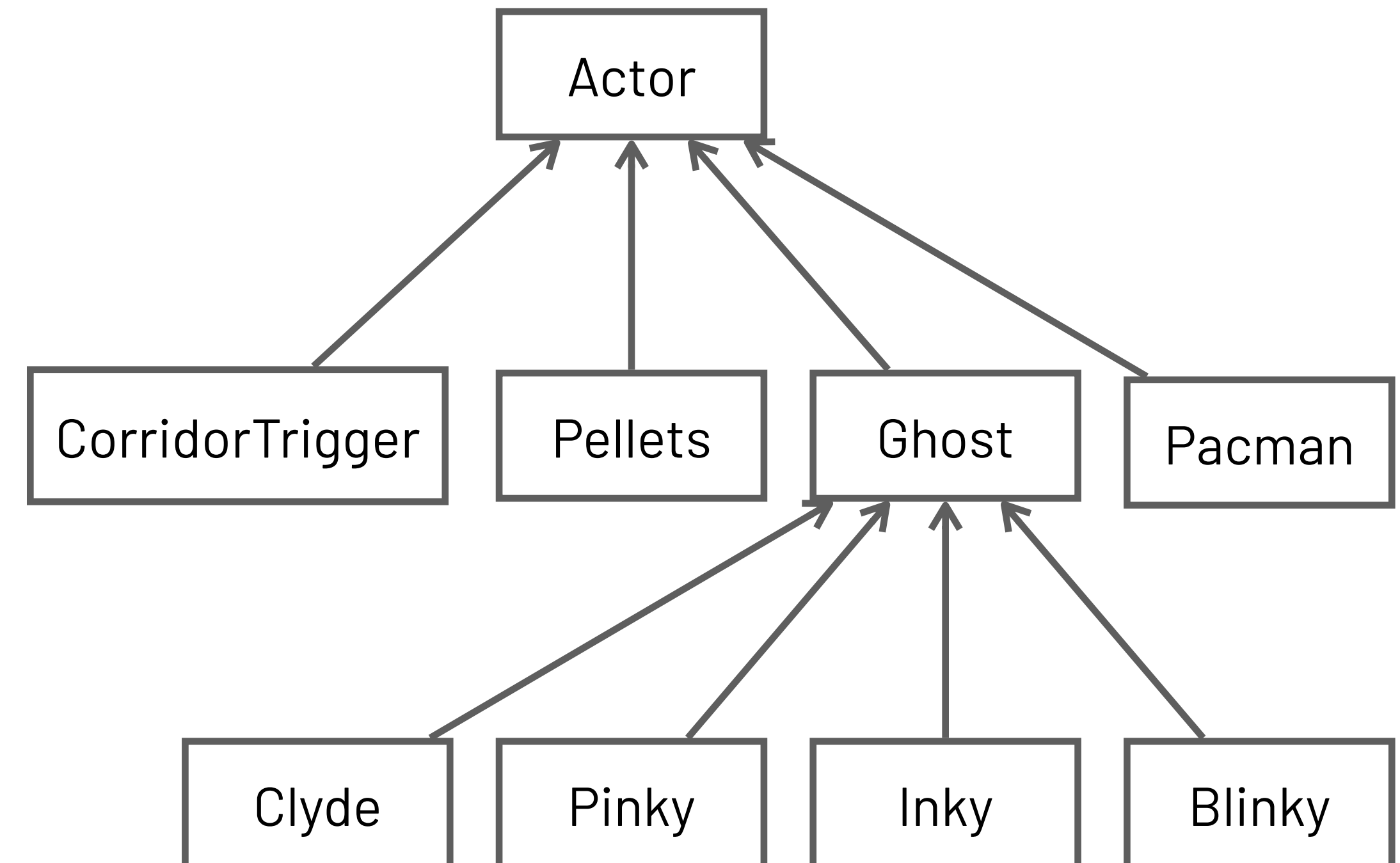
- ▶ Modelo de hierarquia de classes
- ▶ Modelo de componentes
- ▶ Modelo híbrido

# Modelo de Hierarquia de Classes

O comportamento dos objetos do jogo é definido e compartilhado utilizando uma hierarquia de classes, com a raiz em um classe base.



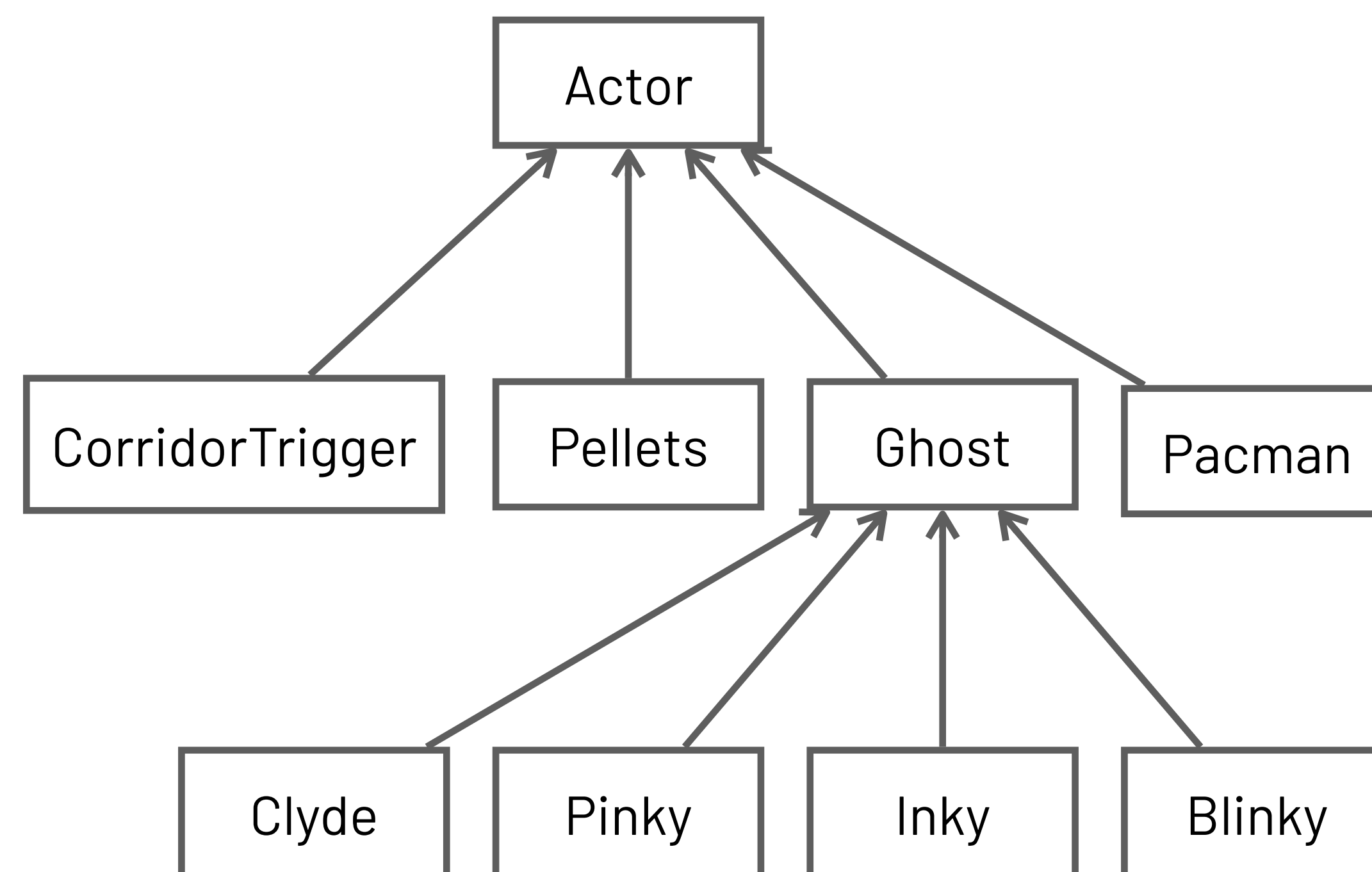
*Pacman - Namco, 1980 (arcade)*



# Problema do Modelo de Hierarquia de Classes

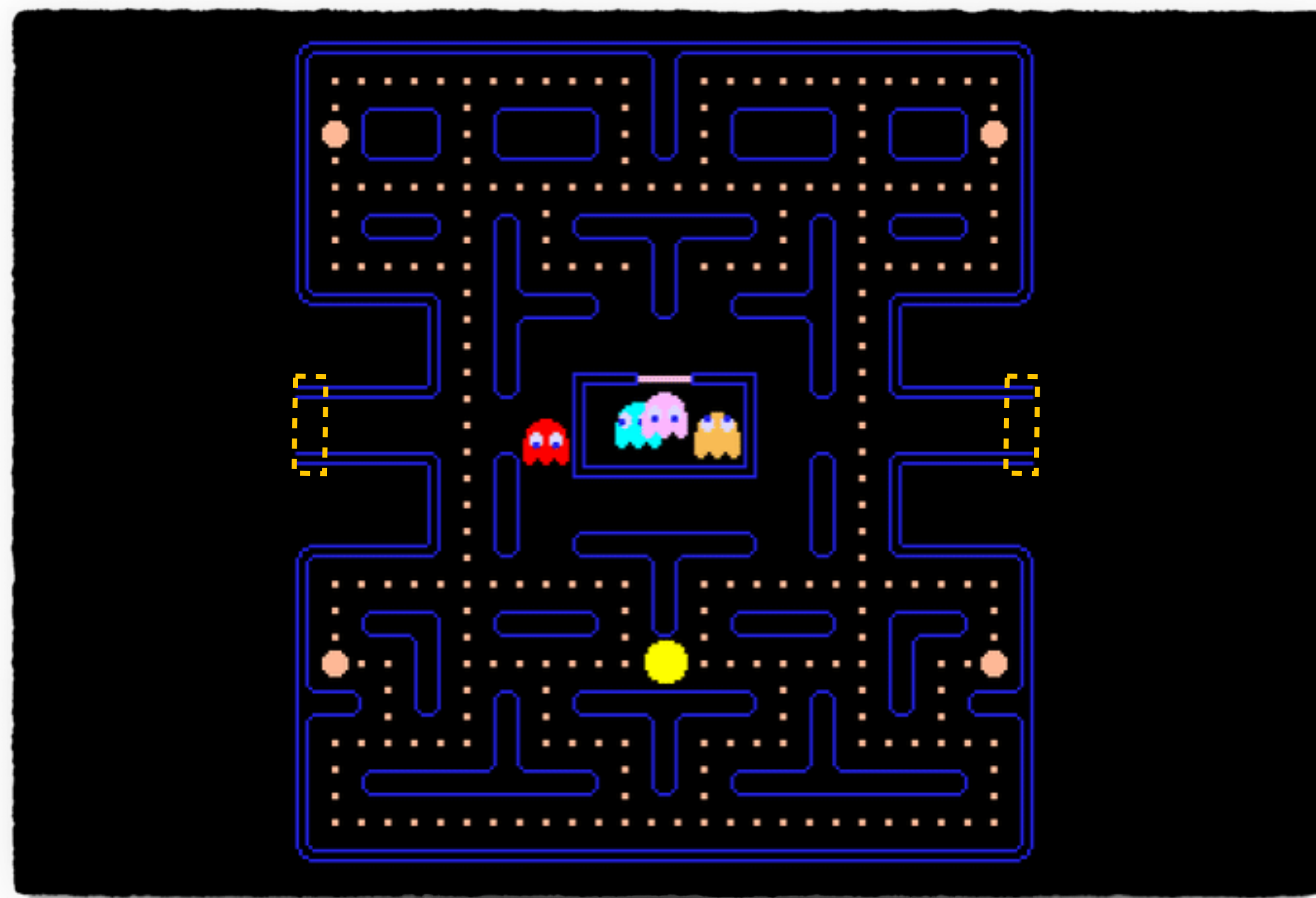
Todo objeto do jogo necessariamente terá todas as propriedades e funções da classe base (Actor), porém:

- ▶ Nem todo objeto precisa de Update ();
- ▶ Nem todo objeto precisa de Draw ();
- ▶ O problema aumenta proporcionalmente com a complexidade do jogo.

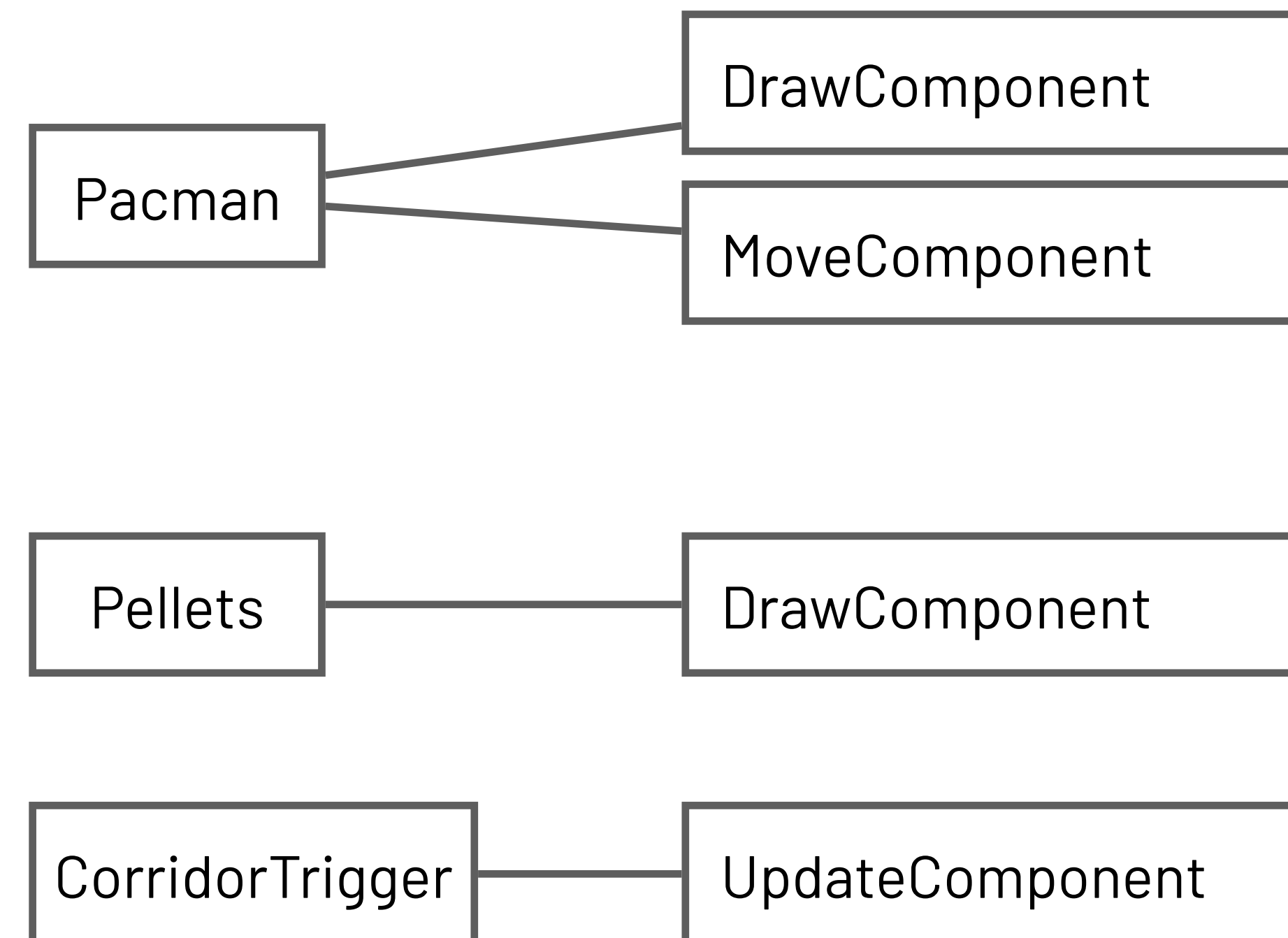


# Modelo de Componentes

Cada objeto do jogo tem uma coleção de componentes que, quando combinados, definem a sua funcionalidade (e.g, Unity).

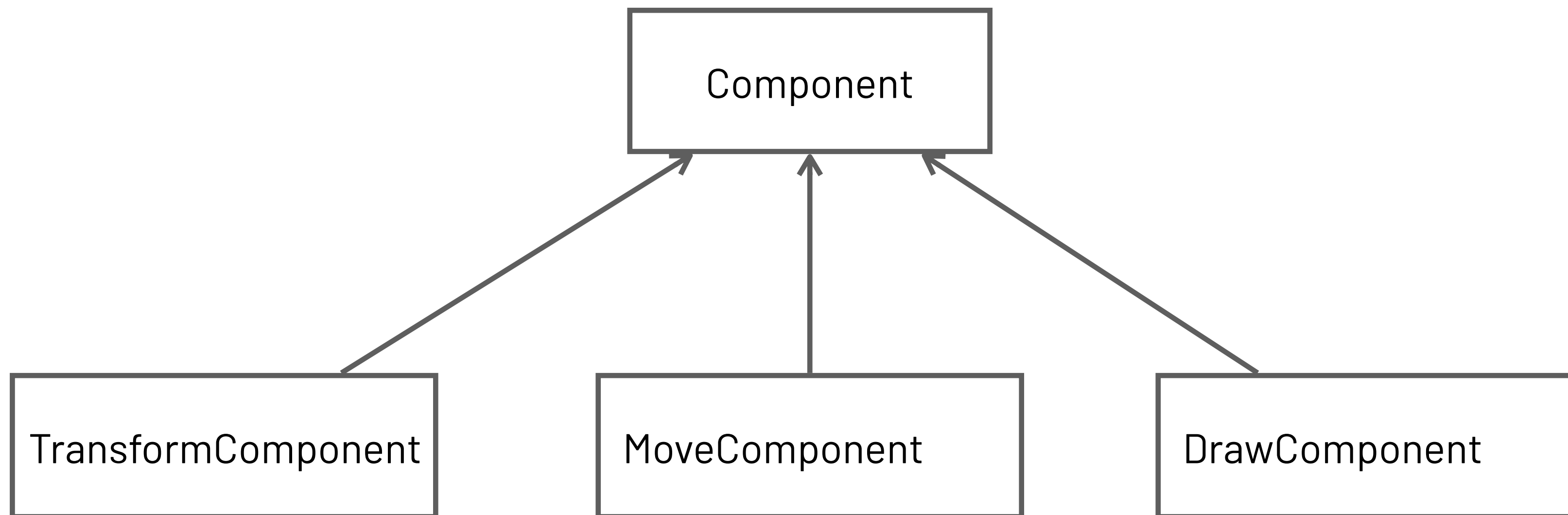


*Pacman - Namco, 1980 (arcade)*



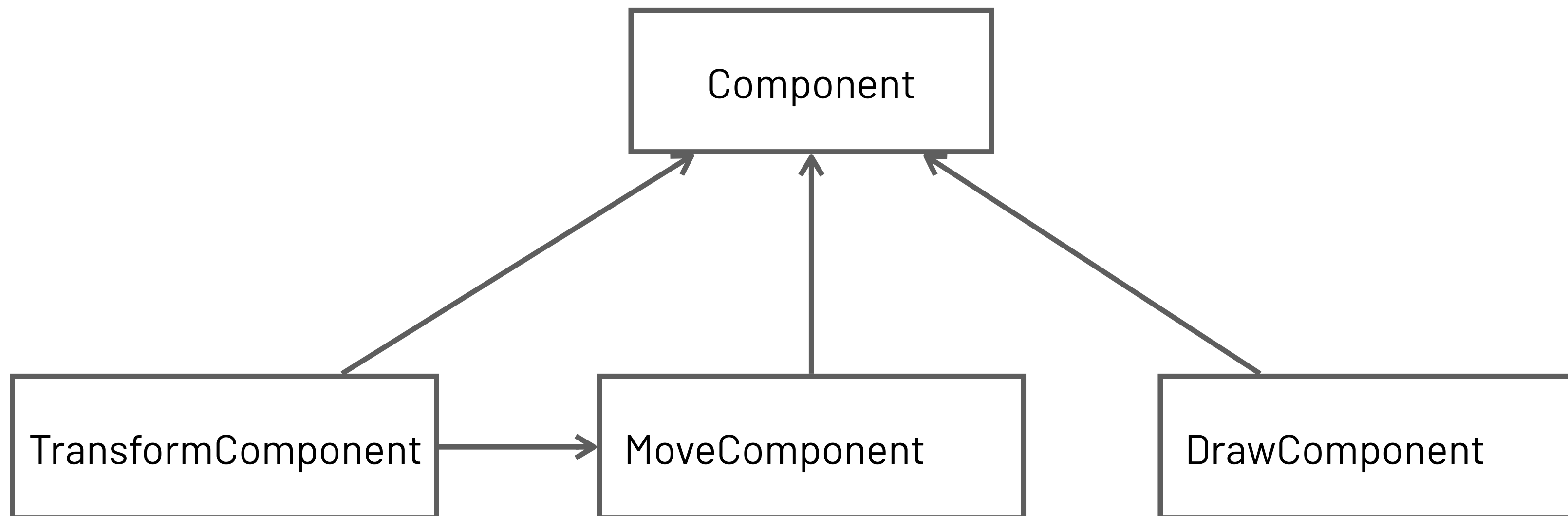
# Modelo de Componentes

Os componentes também formam uma hierarquia de classes, porém muito mais rasa!



# Problemas do Modelo de Componentes

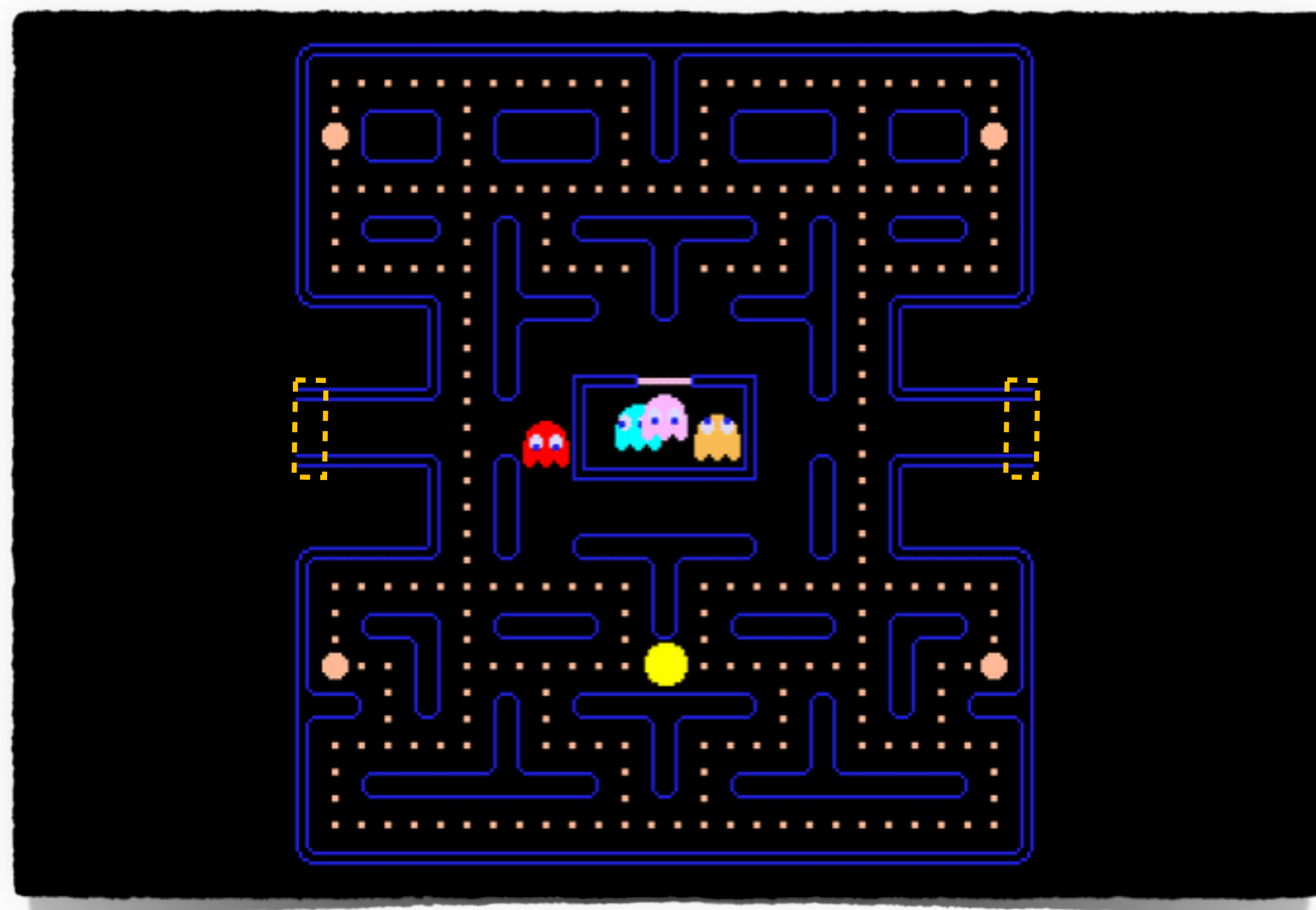
- ▶ Objetos do jogo precisam buscar por componentes em uma lista;
- ▶ Diferentes componentes precisam comunicar entre si.





# Modelo Híbrido

Mistura de hierarquia de classes e componentes (e.g, Unreal).



*Pacman - Namco, 1980 (arcade)*

Actor

Propriedades:

- Transform (posição, rotação, escala)
- Components

Funções:

- GetComponent()

Funções opcionais:

- OnUpdate();
- OnProcessInput();



# Próximas aulas

## **L2:** Pong – Parte 1

Implementar o laço principal do jogo utilizando uma abordagem de taxa de quadros dinâmica.

## **A3:** Álgebra Linear

Álgebra linear para computação gráfica e jogos digitais.