DCC192

2025/2

UFMG

# Desenvolvimento de Jogos Digitais

## A5: Game Loop

Prof. Lucas N. Ferreira

# Logística

## Avisos

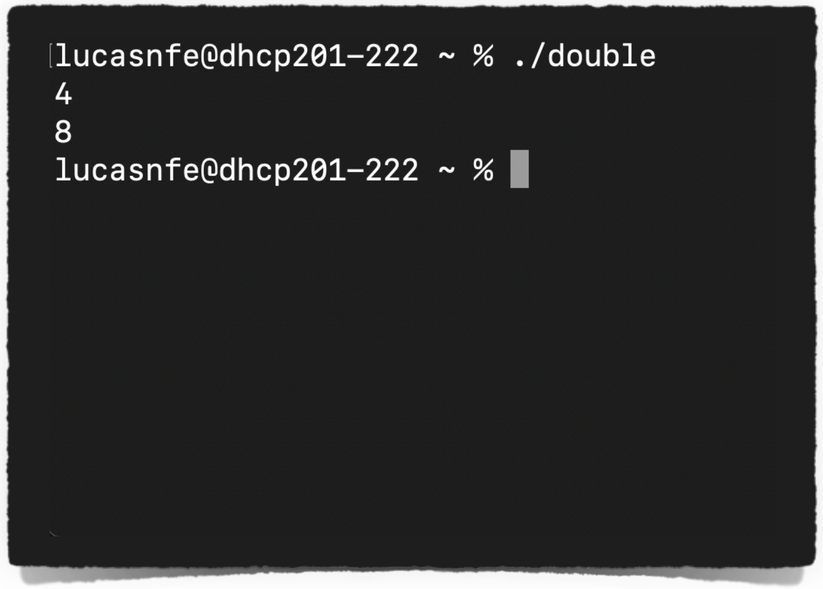
- A entrega do **TP0: Configuração Inicial** é hoje às 11:59h!
- O **TP1: Pong** será disponibilizado até sexta-feira!

# Plano de Aula

- Game Loop
  - Eventos de entrada
  - Atualização de objetos do jogo
  - Geração de saída
- Gerenciamento do tempo do jogo
  - Intervalos de tempo (FPS) fixos
  - Intervalos de tempo (FPS) variáveis
  - Intervalos de tempo (FPS) semi-fixos

# Execução em lote (batch)

Os programas que escrevemos na graduação são geralmente executados em lote (*batch*):

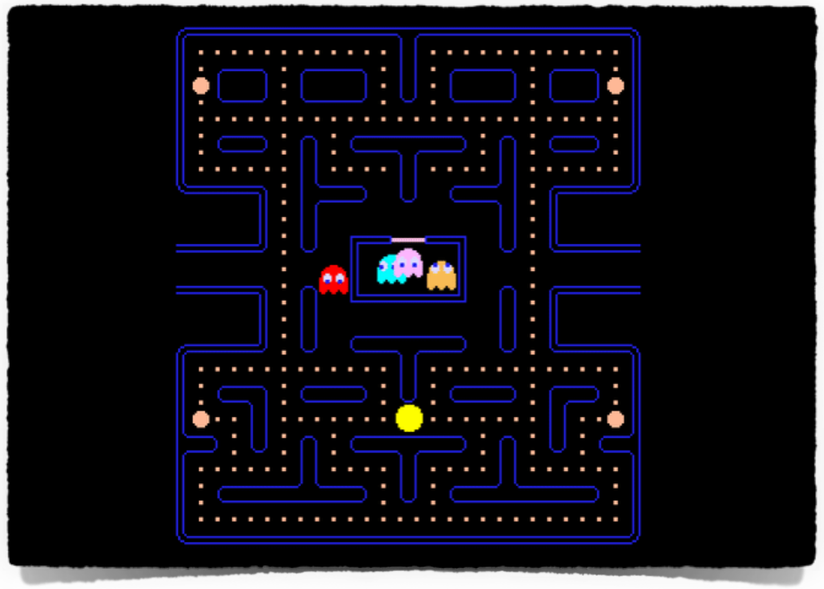



```
1. int x;
2. cin >> x;
3. cout << 2 * x << endl;
```

- Processa dados de uma só vez, sem interação do usuário durante a execução
- Foco em eficiência no processamento de dados

# Game Loop

Jogos digitais são programas interativos em tempo real que executam em loop:



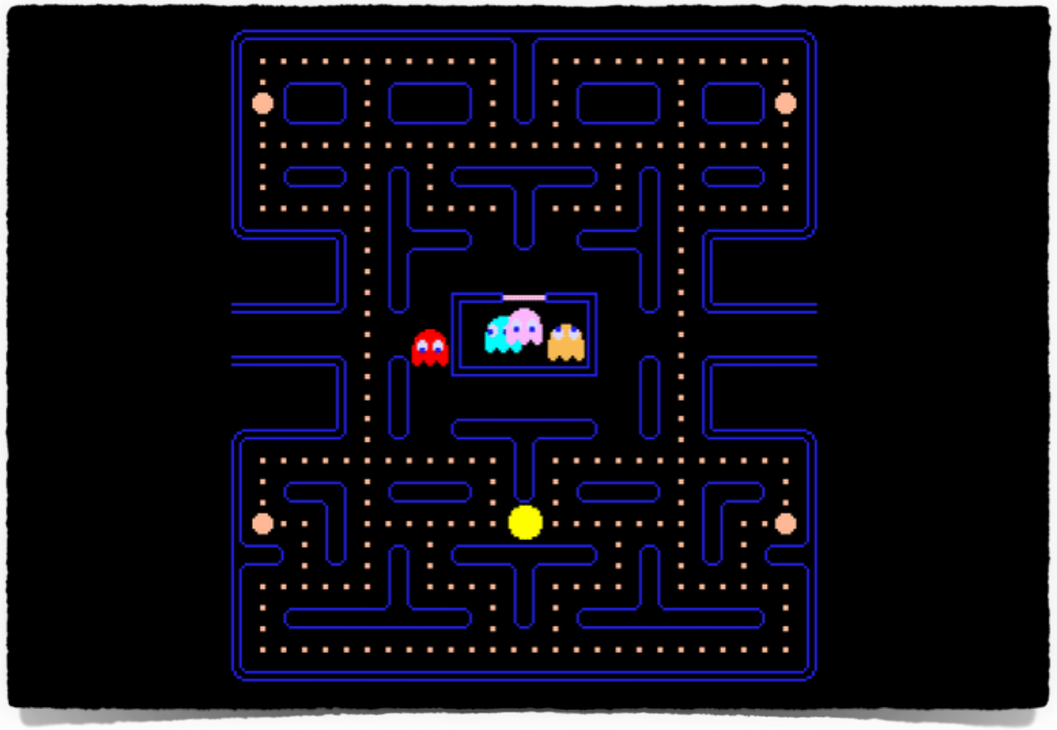
```
1. while Game is running
2.     Process input
3.     Update game world
4.     Generate output
```

*Atualiza estado do jogo mesmo que nenhum evento tenha sido produzido!*
*Ex.: posição dos fantasmas*

- Processa eventos do usuário, atualiza o estado jogo e gera saída constantemente
- Foco em baixa latência e alta taxa de quadros

# Game Loop: Processar Entrada

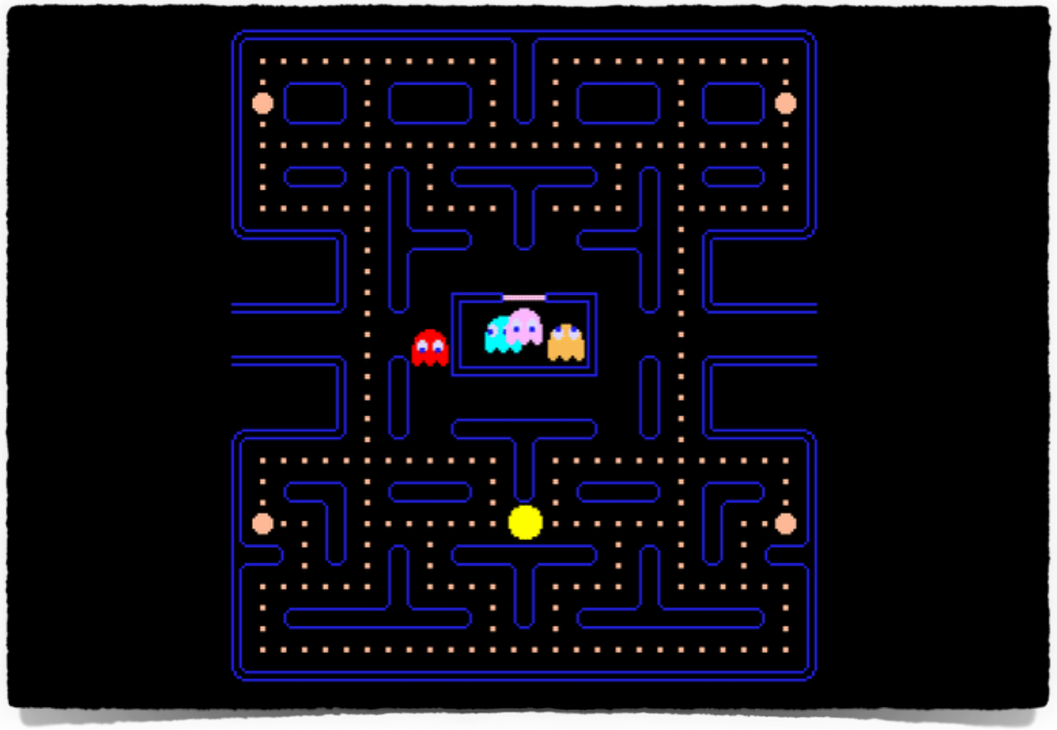
1. A primeira etapa de todo game loop é processar os eventos de entrada do jogador:



```
while Game is running
    ▸ Process input
        ▸ ⌨️ Keyboard
        ▸ 🖱️ Mouse
        ▸ 🕹️ Joystick
        ▸ ...?
    Update game world
    Generate output
```

# Game Loop: Atualizar Objetos do Jogo

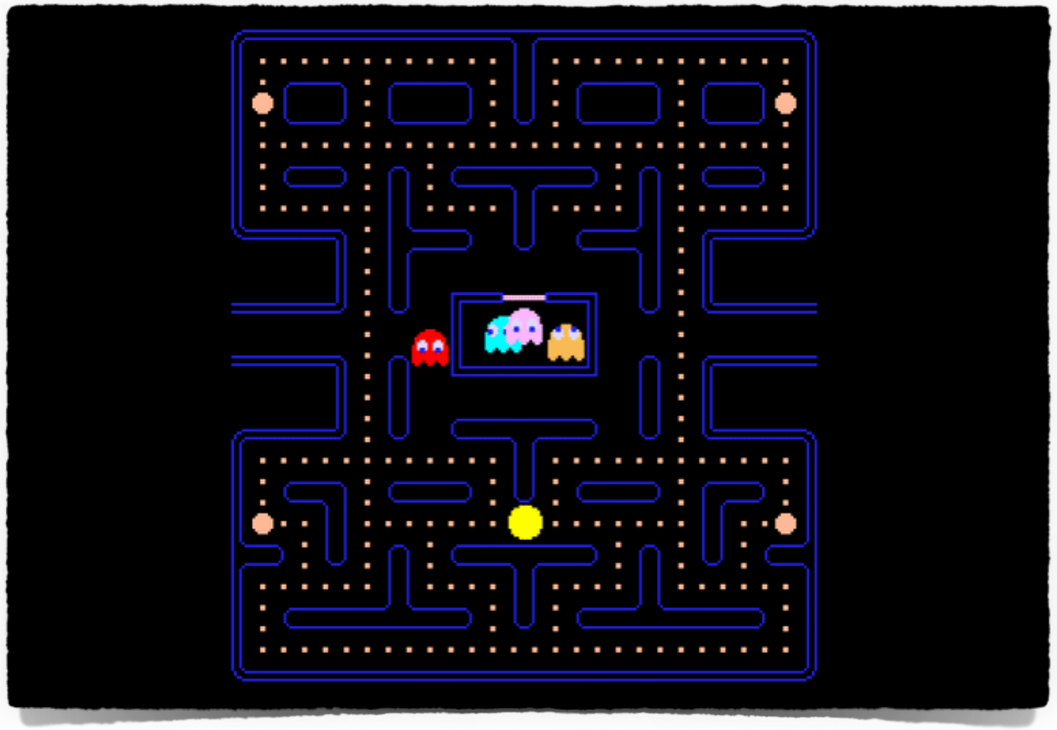
2. A segunda etapa é a **atualização do estado de todos os objetos do jogo**:



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

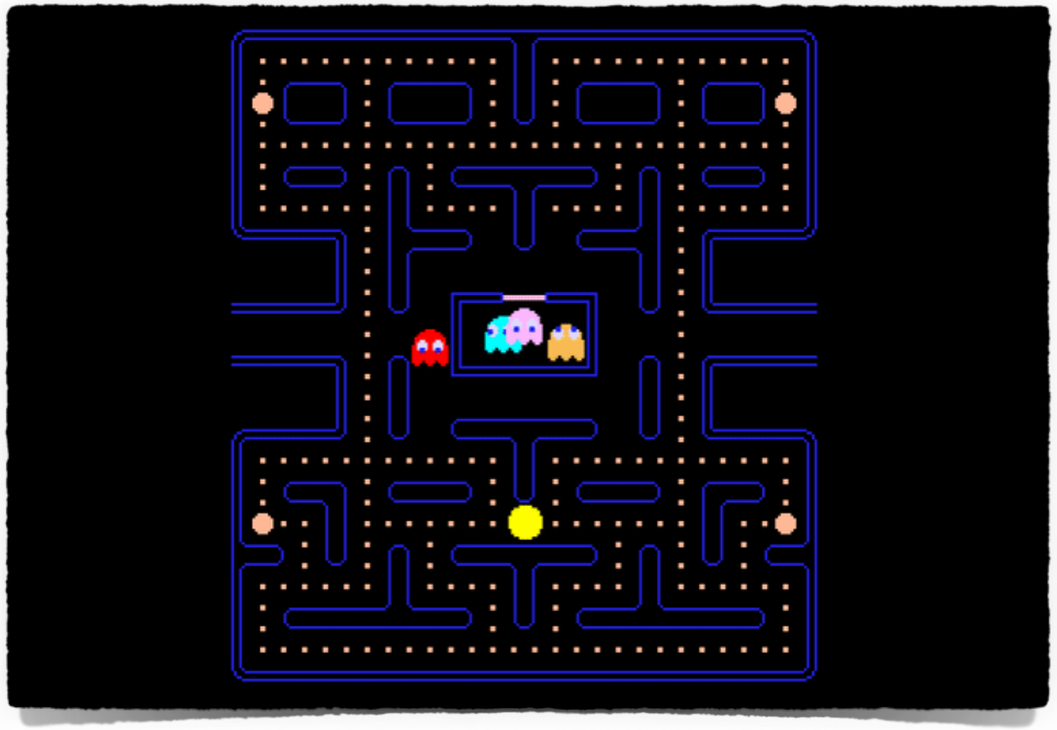
# Game Loop: Gerar Saídas

3. A terceira etapa é **gerar saídas** com base no estado atualizado dos objetos do mundo:



```
while Game is running
        Process input
        Update game world
        ▸ Generate output
            ▸ Image
            ▸ Sound
            ▸ ...?
```

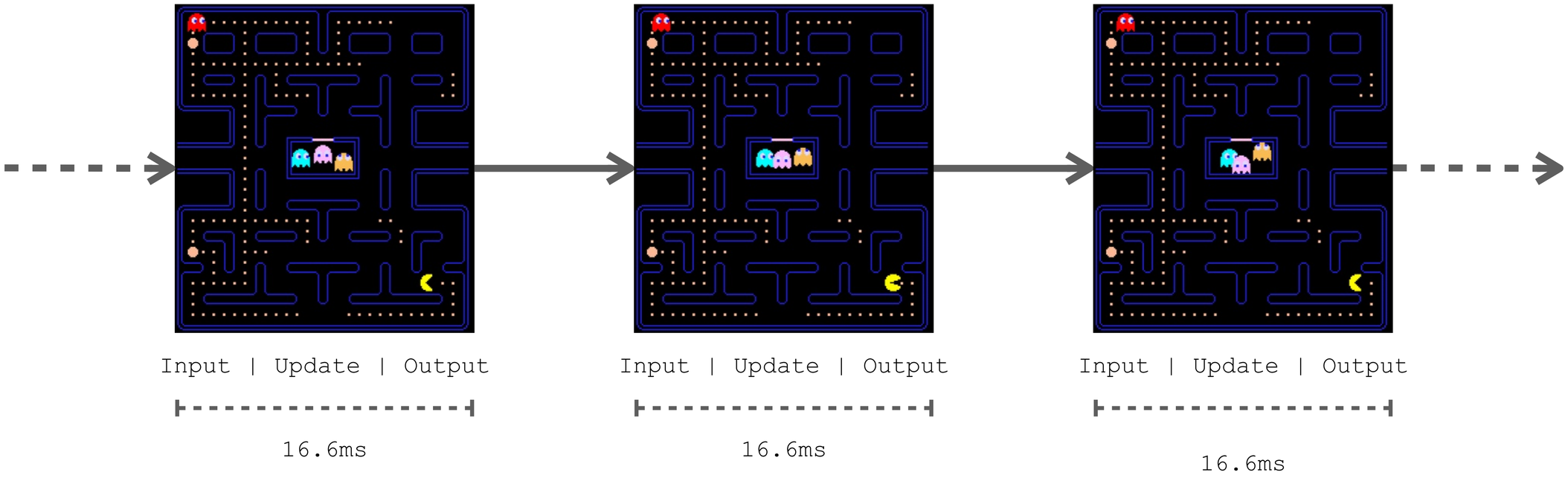
# Pseudocódigo do PacMan



```
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. jogos

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




Por exemplo: 60 FPS, 1/60 = 16.6ms por quadro

# Tempo real vs. tempo do jogo

O **tempo do jogo** pode ser diferente do **tempo de relógio**:

*Tempo decorrido no jogo*

*Tempo  decorrido no mundo real*

# Tempo real vs. tempo do jogo

O tempo do jogo pode ser diferente do tempo de relógio:

*Tempo decorrido no jogo*

*Tempo  decorrido no mundo real*

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


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

# Tempo real vs. tempo do jogo

O tempo do jogo pode ser diferente do tempo de relógio:

*Tempo decorrido no jogo*

*Tempo  decorrido no mundo real*

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


*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.*

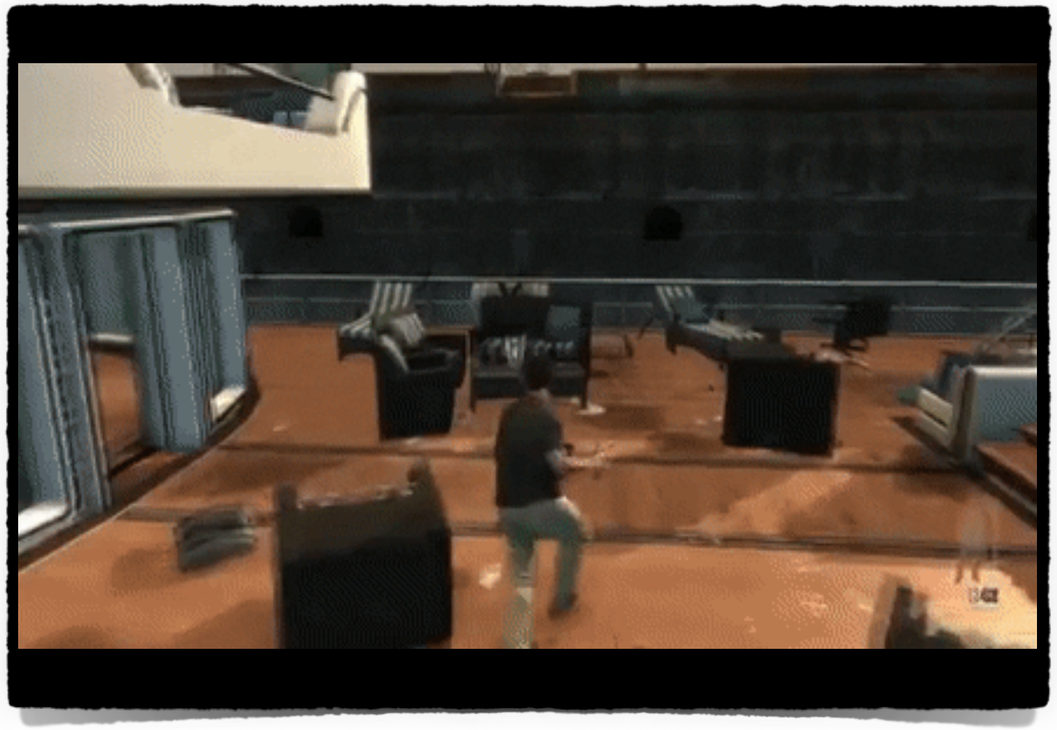
# Tempo real vs. tempo do jogo

O tempo do jogo pode ser diferente do tempo de relógio:

*Tempo decorrido no jogo*

*Tempo  decorrido no mundo real*

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



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

# Tempo real vs. tempo do jogo

O tempo do jogo pode ser diferente do tempo de relógio:

*Tempo decorrido no jogo*

*Tempo  decorrido no mundo real*

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

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

# Game Loop e Tempo de Jogo

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

**Qual o problema com esse loop?**
Nenhum controle sobre o tempo!

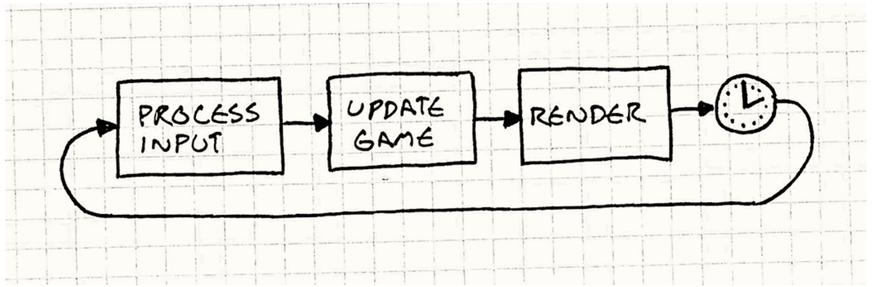
```
enemy.position += 5;
```

- Processador 8 MHz → jogador vai mover 1x
- Processador 16 MHz → jogador vai mover 2x

Existem três técnicas principais para gerênciamento do tempo do jogo:

- Intervalos de tempo (FPS) fixos
- Intervalos de tempo (FPS) variáveis
- Intervalos de tempo (FPS) semi-fixos

# Intervalos de tempo fixos




```
MS_PER_FRAME = 16.6
while (gameIsRunning) {
        double start = getCurrentTime();
        ProcessInput();
        Update();
        GenerateOutput();
        Sleep(start + MS_PER_FRAME - getCurrentTime())
}
```

**1. Definir um *frame time* ideal (ex. 1/60 milisegundos)**

**2. Marcar o tempo no início do quadro**

**3. Dormir no tempo que sobrou!**

**Qual o problema com esse loop?**
Se o jogo rodar muito devagar, o tempo de dormir fica negativo.

# Intervalos de tempo variáveis

```
double lastTime = getCurrentTime();
-------------------------------------------------------------------------------
while (gameIsRunning) {                                  1. Inicialiar o tempo do último  quadro

        double current = getCurrentTime();

        double deltaTime = current - lastTime;
        -------------------------------------------------------------------------------
        ProcessInput();                  2. Calcular quanto tempo (s) passou desde o último quadro

        Update(deltaTime);
        -------------------------------------------------------------------------------
        GenerateOutput();                3. Atualizar objetos em função do delta time

        lastTime = current;
        -------------------------------------------------------------------------------
}                                                        4. Atualizar o tempo do último quadro

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

# Intervalos de tempo 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 ~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 ~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!**

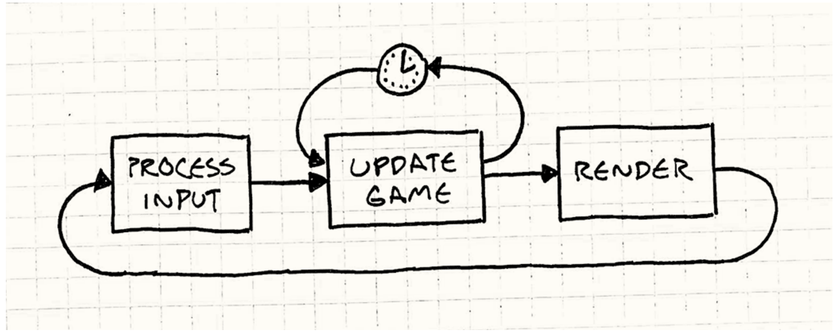
# Intervalos de tempo variáveis

```
double lastTime = getCurrentTime();
while (gameIsRunning) {
        double current = getCurrentTime();
        double deltaTime = current - lastTime;
        ProcessInput();
        Update(deltaTime);
        GenerateOutput();
        lastTime = current;
}
```

**Qual o problema com delta time loop?**

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

# Intervalos de tempo semi-fixos




```
double lastTime = getCurrentTime();
double lag = 0.0;
MS_PER_FRAME = 16.6;
while (gameIsRunning) {
        double current = getCurrentTime();
        deltaTime = (current - lastTime)/1000;
        lastTime = current;
        lag += deltaTime;
        ProcessInput();
        while(lag >= MS_PER_FRAME) {
          Update();
          lag -= MS_PER_FRAME;
        }
        GenerateOutput();
}
```

**1. Calcular quanto tempo (s) passou desde o último quadro**

**2. Atualizar os quadros com FPS fixo pelo tempo que tivermos disponível**

**Obs: `GerenrateOutput()` e `ProcessInput()` podem executar o mais rápido que puderem**

# Próxima aula

## A6: Game Objects

- Atualizando Objetos do Jogo
- Objetos Dinâmicos , Estáticos e Gatilhos
- Modelagem de Objetos
  - Modelo de hierarquia de classes
  - Modelo de componentes
  - Modelo híbrido