LOOP x TEMPO x VELOCIDADE

Alysson Diniz dos Santos

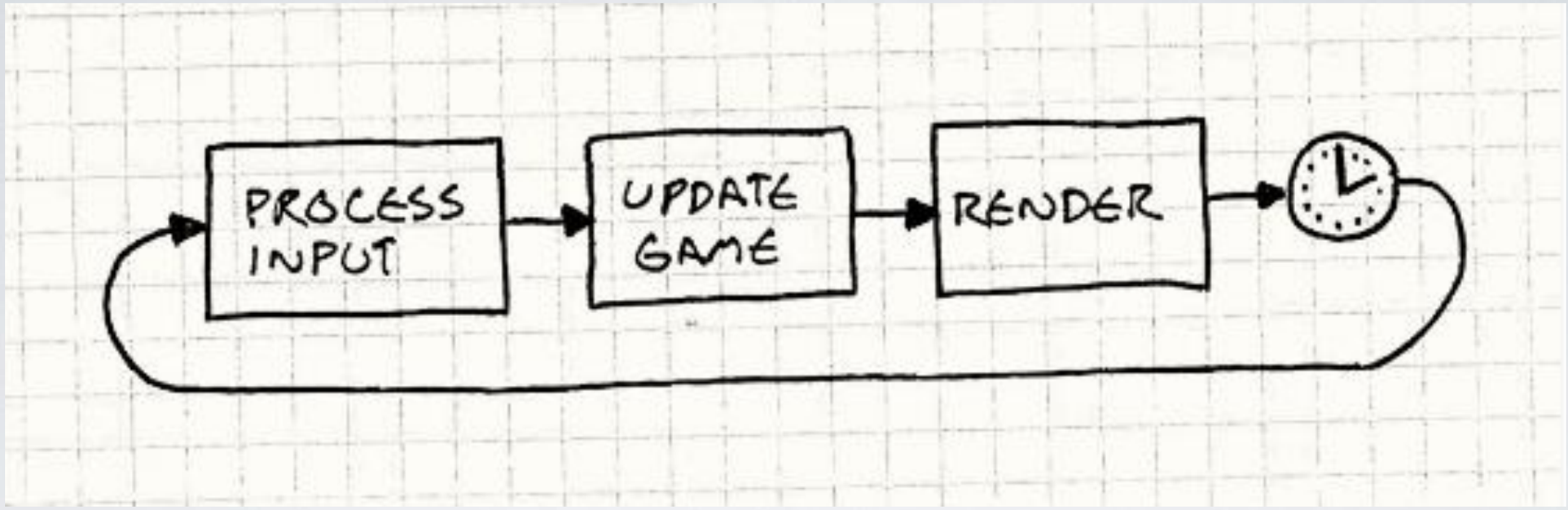
Na aula passada

Loop de uma aplicaçao

```
while (true)
{
   Event* event = waitForEvent();
   dispatchEvent(event);
}
```

Loop de jogo a tempo fixo

```
while (true)
{
   processInput();
   update();
   render();
}
```

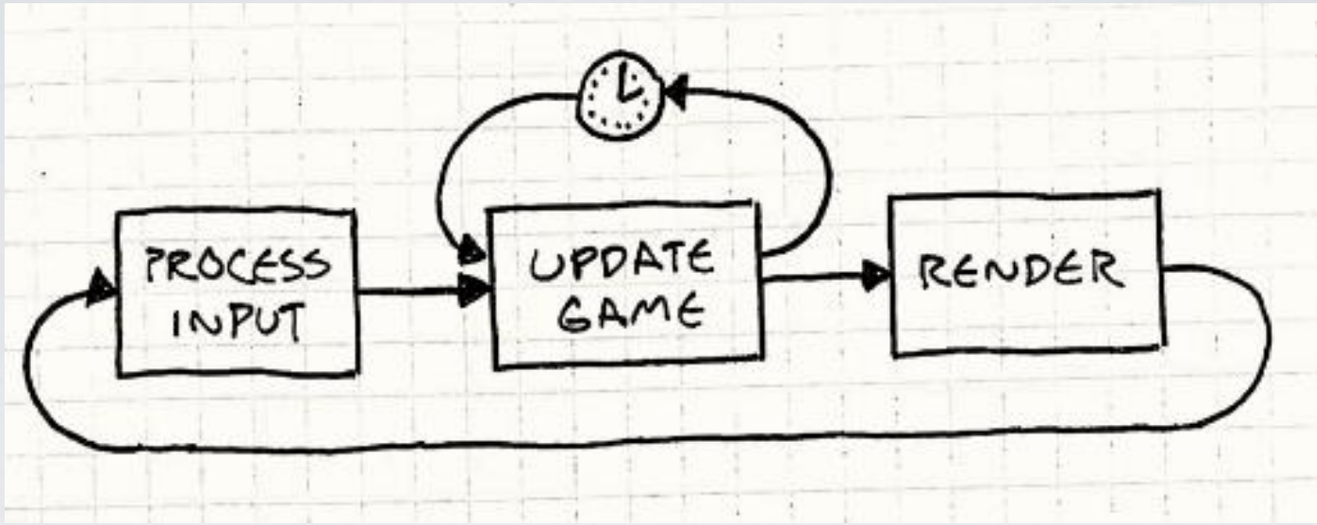



```
while (true)
{
  double start = getCurrentTime();
  processInput();
  update();
  render();

  sleep(start + MS_PER_FRAME - getCurrentTime());
}
```

Tempo fixo, com sincronizaçao

```
double lastTime = getCurrentTime();
while (true)
{
   double current = getCurrentTime();
   double elapsed = current - lastTime;
   processInput();
   update(elapsed);
   render();
   lastTime = current;
}
```

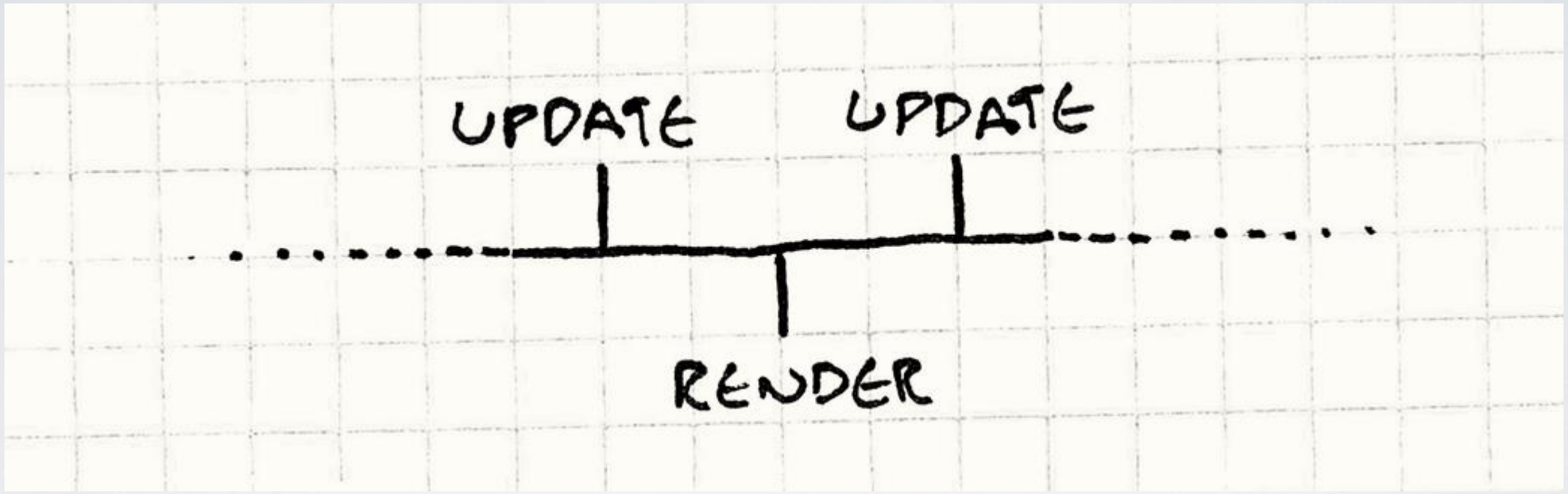
Tempo variavel




```
double previous = getCurrentTime();
double lag = 0.0;
while (true)
{
  double current = getCurrentTime();
  double elapsed = current - previous;
  previous = current;
  lag += elapsed;

  processInput();

  while (lag >= MS_PER_UPDATE)
  {
    update();
    lag -= MS_PER_UPDATE;
  }

  render();
}
```

Tempo de update fixo, tempo de renderizaçao variavel

Um ultimo problema:



Soluçao: adicionar o lag ao render

```
render(lag / MS_PER_UPDATE);
```

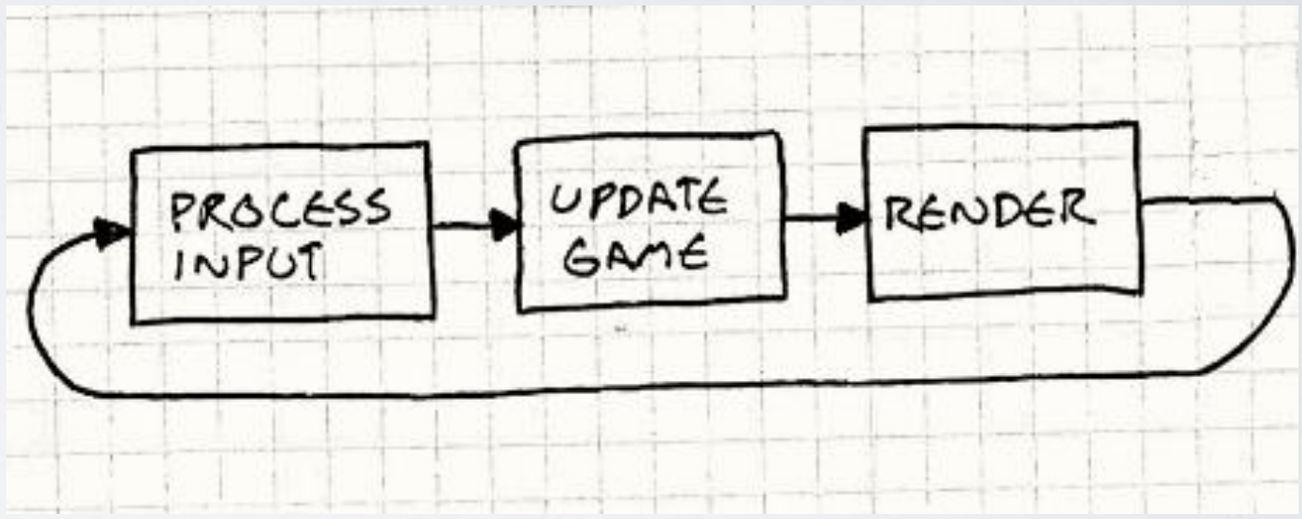
Atualmente, qual o tipo de loop de nosso jogo?

Como implementar um loop a tempo fixo (sem sincronizaçao) no javascript?

```
while(true) {
    move()
    draw()
}
```

Como o javascript executa em uma thread unica, essa abordagem muito provavelmente faria o script quebrar

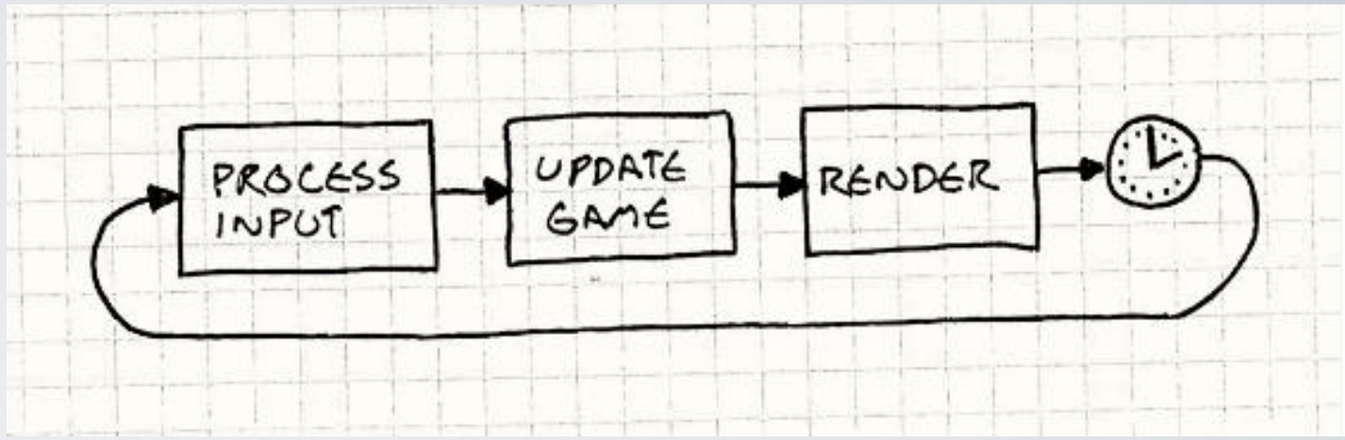
requestAnimationFrame() – a funçao passada como parametro é invocada tao cedo quando o browser esteja pronto para atualizar a tela.




```
function mainLoop() {
    move();
    draw();
    requestAnimationFrame(mainLoop);
}
...
requestAnimationFrame(mainLoop);
```

Introduzindo sincronizaçao

```
var lastFrameTimeMs = 0;
var maxFPS = 60;
var timeStep = 1000/maxFPS;
```



...

```
function mainLoop(timeStamp) {
    if(timeStamp < lastFrameTimeMs + timeStep) {
        requestAnimationFrame(mainLoop);
        return;
    }
    lastFrameTimeMs = timeStamp;
    move();
    draw();
    requestAnimationFrame(mainLoop);
}
```

Ligando a aplicaçao ao tempo real

ballX += ballSpeedX;

Se ballSpeedX = 5, isso quer dizer que a bolinha se moverá a **5 pixels/frame**.

Logo, se o jogo roda a 30fps, a bolinha se movera **150 pixels em 1 segundo**.

E a 60fps, **300 pixels em 1 segundo**.

ballX += ballSpeedX*deltaTime;

Se ballSpeedX = 150, isso quer dizer que a bolinha se movera 150 **pixels/segundo**.

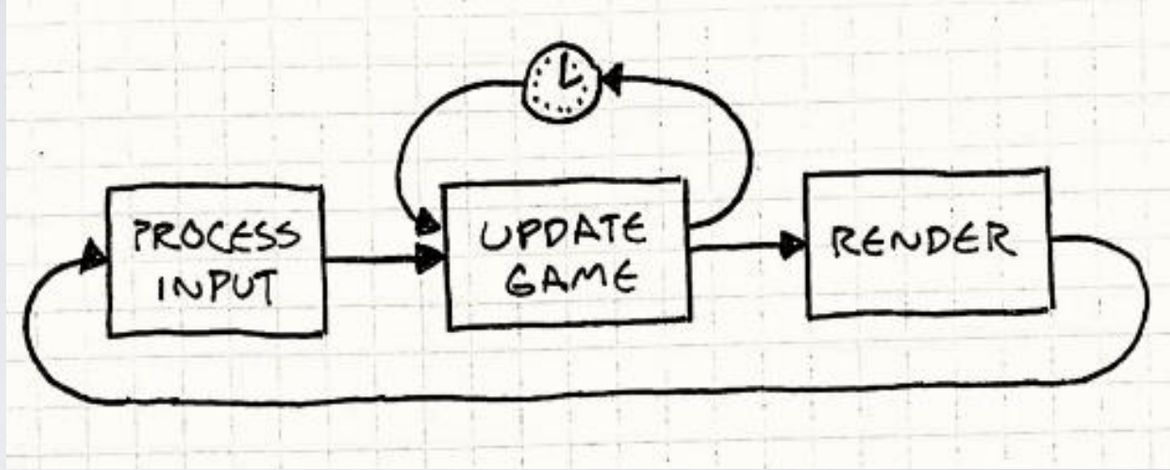
Se o jogo roda a 30fps, deltaTime será igual a 0,033 (1/30).

Logo, a bolinha se movera a **5 pixels por frame** (150*0,033)

Se o jogo roda a 60fps, deltaTime será igual a 0,016 (1/60).

Logo, a bolinha se movera a **2.5 pixels por frame** (150*0,016)

```
var delta = 0;

...

function mainLoop(timeStamp) {
    ...
    delta = timeStamp – lastFrameTimeMs;
    lastFrameTimeMs = timeStamp;
    move(delta);
    ...
}

function move(delta) {
    ...
    ballX += ballSpeedX*delta;
    ballY += ballSpeedY*delta;
}
```




```
function mainLoop(timeStamp) {
    ...
    //delta agora acumula o tempo que ainda não foi simulado
    delta += timeStamp – lastFrameTimeMs;
    lastFrameTimeMs = timeStamp;
    //E o laco simula a passagem do tempo em passos fixos
    while(delta >= timeStep) {
        move(timeStep);
        delta -= timeStep;
    }
    draw();
    requestAnimationFrame(mainLoop)
}
```

Se o valor em delta crescer muito (o que pode ocorrer graças a um excesso de processamento, ou ao jogo ter ficado aberto em uma aba secundaria, por exemplo), o jogo pode acrescentar comportamento erratico (crashar).

Uma soluçao possivel seria oferecer uma rota de escape:

```
var numUpdateSteps = 0;
while (delta >= timestep) {
    update(timestep);
    delta -= timestep;
    // Sanity check
    if (++numUpdateSteps >= 240) {
        panic(); // fix things
        break; // bail out
    }
}
```

O que fazer na funçao de panico: DEPENDE

- Em um jogo multiplayer baseado em turnos, por exemplo, todos os jogadores experienciam o jogo na velocidade do jogador mais lento.

- Caso um jogador esteja **muito** lento, ele é excluido da sessao.

O que fazer na funçao de panico: DEPENDE

- Em um fps multiplayer o servidor guarda o **estado autoritario** do jogo (authoritative state)

- Caso um jogador esteja muito atras dos outros, ele é catapultado para o estagio autoritario da sessao.

  - Neste caso, o delta é zerado no cliente (o que introduz algum nao-determinismo)

O que fazer na funçao de panico: DEPENDE

- Em um jogo single player pode ser dado algum tempo extra para que o jogo se recupere

- Ou o delta pode ser mesmo zerado (mesmo correndo o risco do nao-determinismo)

Para prevenir, pode ajudar um **controle de FPS**

- Caso o FPS caia muito (antes de ser chamada a funçao de panico), pode-se:
  - Diminuir a qualidade visual
  - Parar atividades fora do loop principal (e.g. sons e tratamento de eventos)
  - Aumentar o time step
  - ...

Referencias:

- Game Programming Patterns – Game Loop (http://gameprogrammingpatterns.com/game-loop.html)
- Fix your timestep (Glenn Fiedler) - https://gafferongames.com/post/fix_your_timestep/