

DCC192

2025/1

UF  G

Desenvolvimento de Jogos Digitais

A10: Gráficos 2D

Prof. Lucas N. Ferreira

Plano de aula



- ▶ Sprites
- ▶ Spritesheets
- ▶ Animações
- ▶ Tilemaps
- ▶ Rolagem de Câmera
- ▶ Efeito de Paralaxe

Monitores



Monitores são compostos por uma matriz de pixels independentes utilizados para formar **imagens** completas:



Monitores CRT

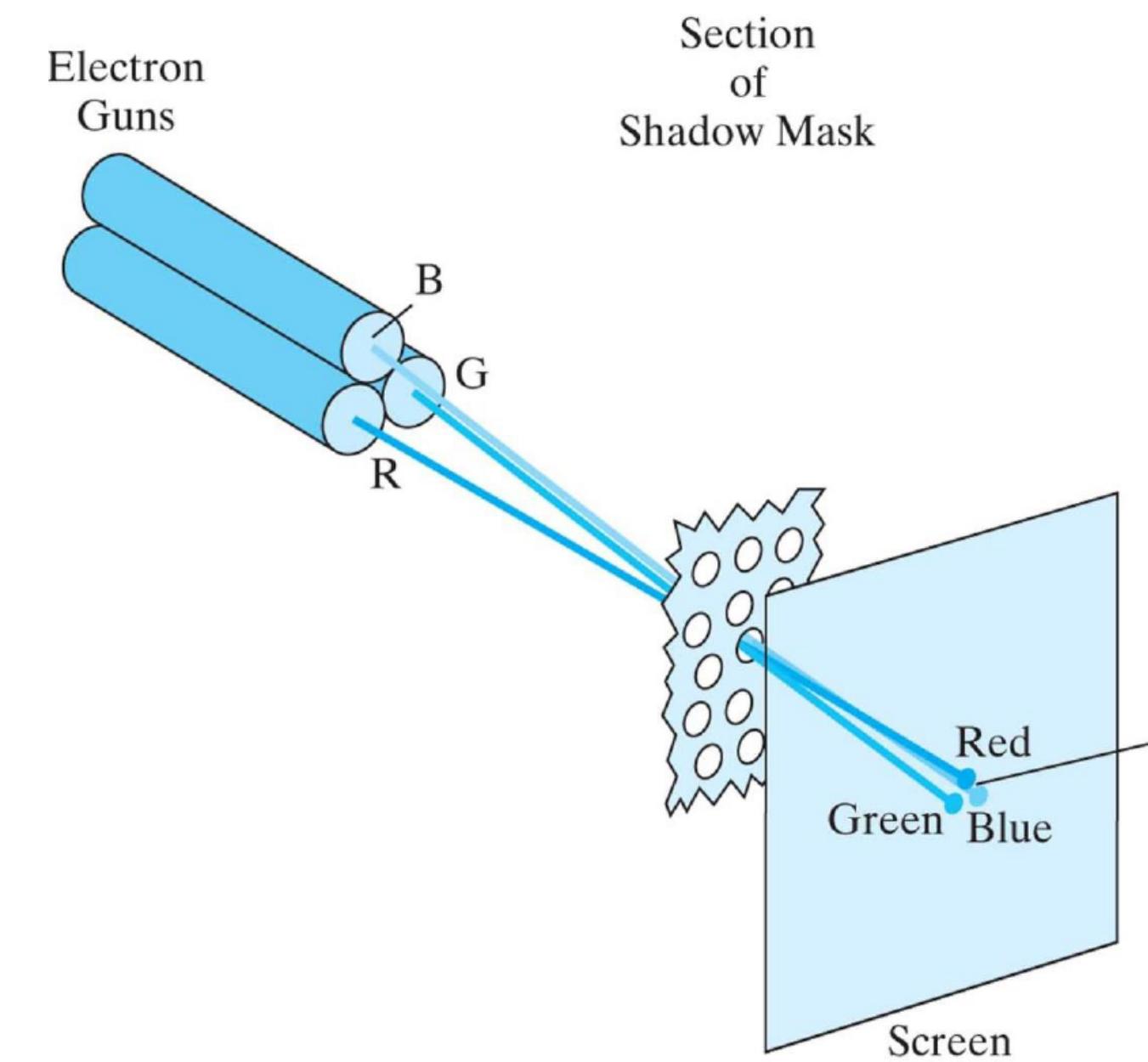
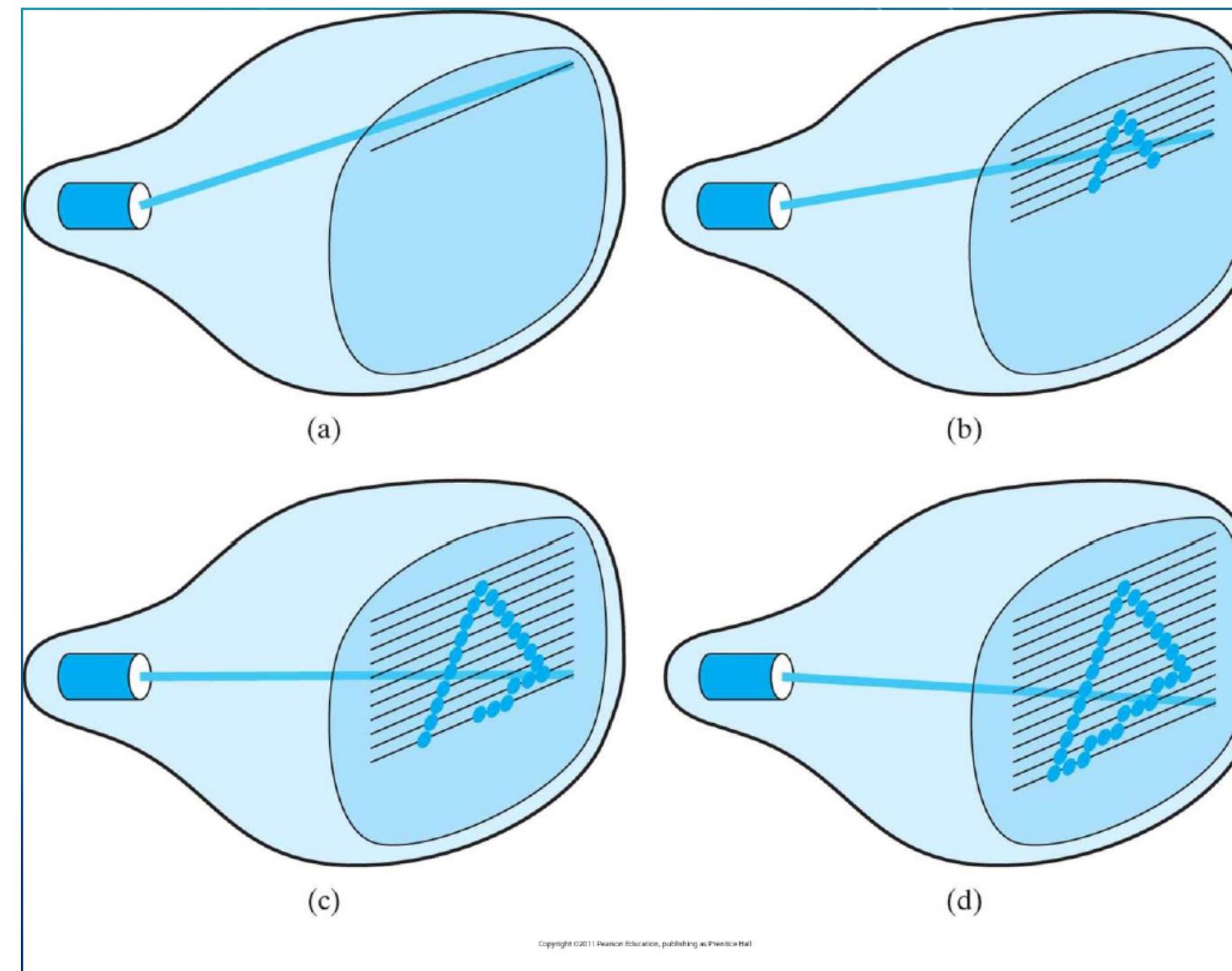


Monitores LED

Monitores CRT (Cathodic Ray Tube)

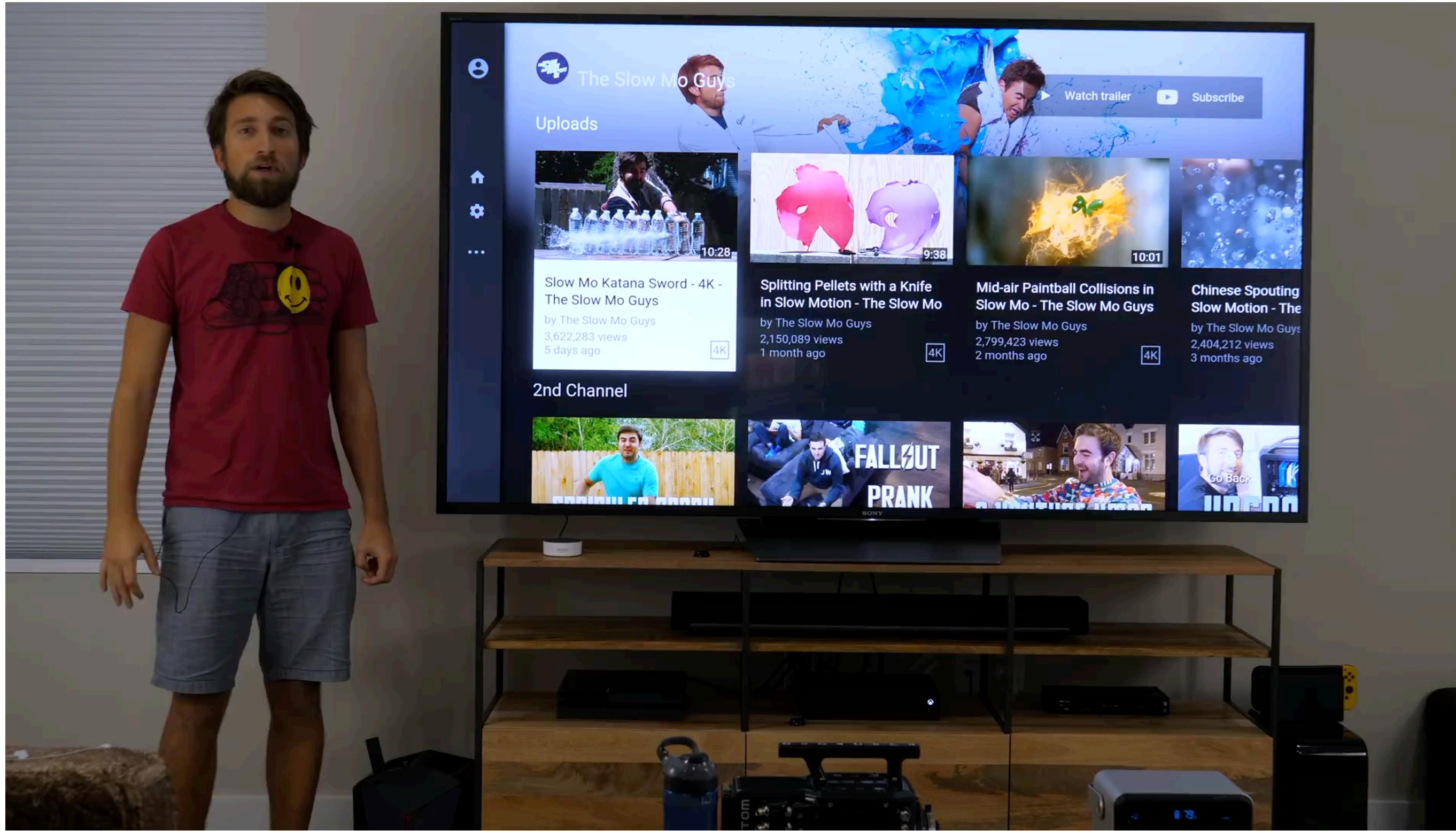


Monitores CRT antigos desenham pixels por meio de um canhão de elétrons que dispara feixes em uma tela revestida de fósforo.



Monitores LED

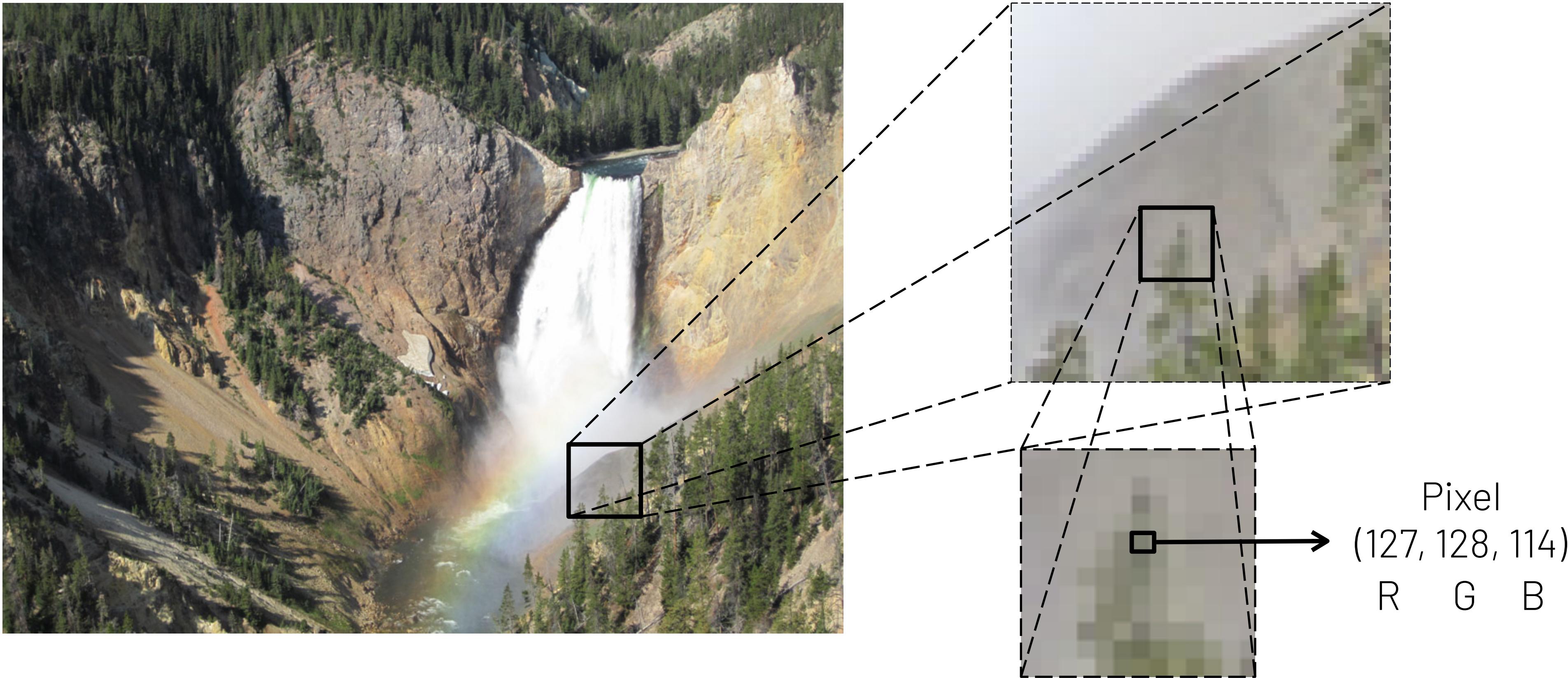
Monitores de LED mais recentes usam lâmpadas de LED independentes para representar os pixels.



https://www.youtube.com/watch?v=3BJU2drirtCM&ab_channel=TheSlowMoGuys

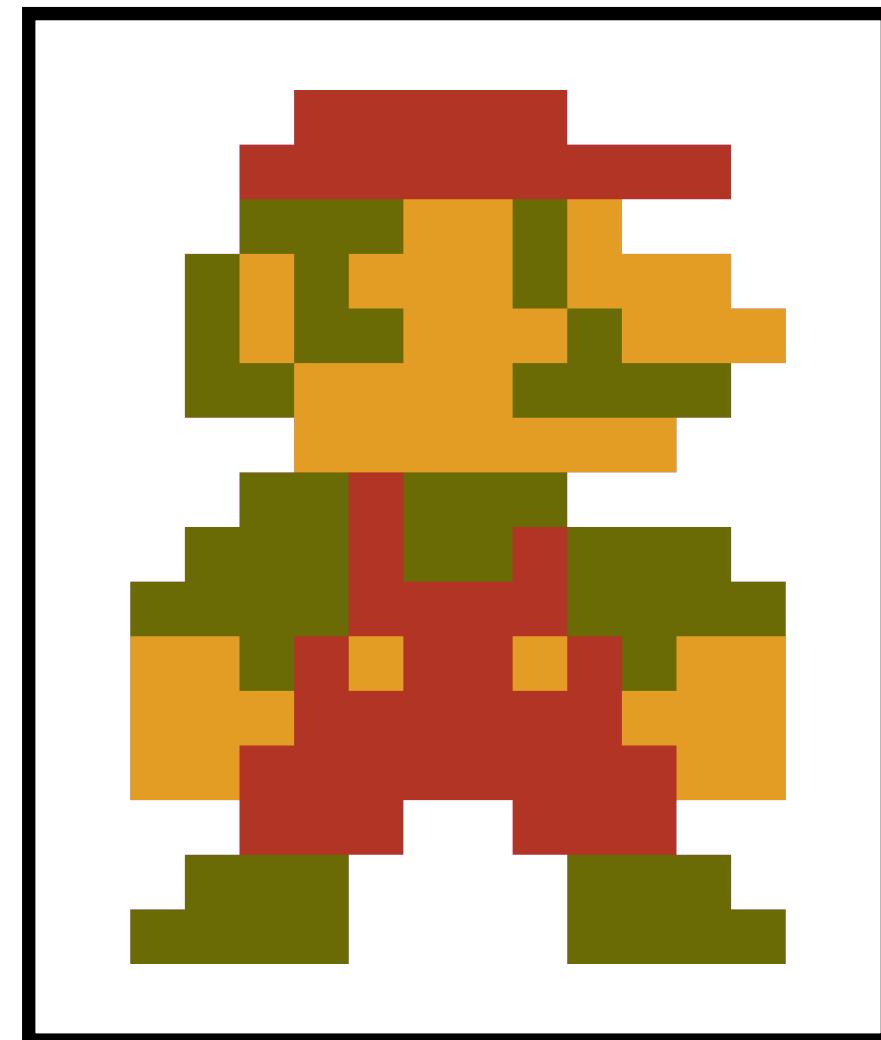
Imagens

Imagens são arranjos bidimensionais de pixels, onde cada pixel é representado por 3 valores (canais): vermelho (R), verde (G) e azul (B)



Sprites

Um **sprite** é uma imagem (i.e., arranjo bidimensional de pixels) utilizada representação visual em 2D de um objeto do jogo (ex. Mario).



```
class Sprite {  
    Vector2 position;  
    int drawOrder;  
    SDL_Texture *texture;  
    void Draw();  
}
```

Carregando Sprites em SDL



Para carregar imagens em SDL, precisamos usar uma biblioteca adicional `SDL_``Image`.`h`

```
#include <SDL2/SDL_image.h>

int main(int argc, char* argv[]) {
    ...

    if (IMG_Init(IMG_INIT_PNG) == 0) {
        std::cerr << "Erro ao inicializar SDL_image: " << IMG_GetError() << std::endl;
        return 1;
    }

    SDL_Texture* texture = IMG_LoadTexture(renderer, "minha_textura.png");

    if (!texture) {
        std::cerr << "Erro ao carregar textura: " << IMG_GetError() << std::endl;
    }

    ...

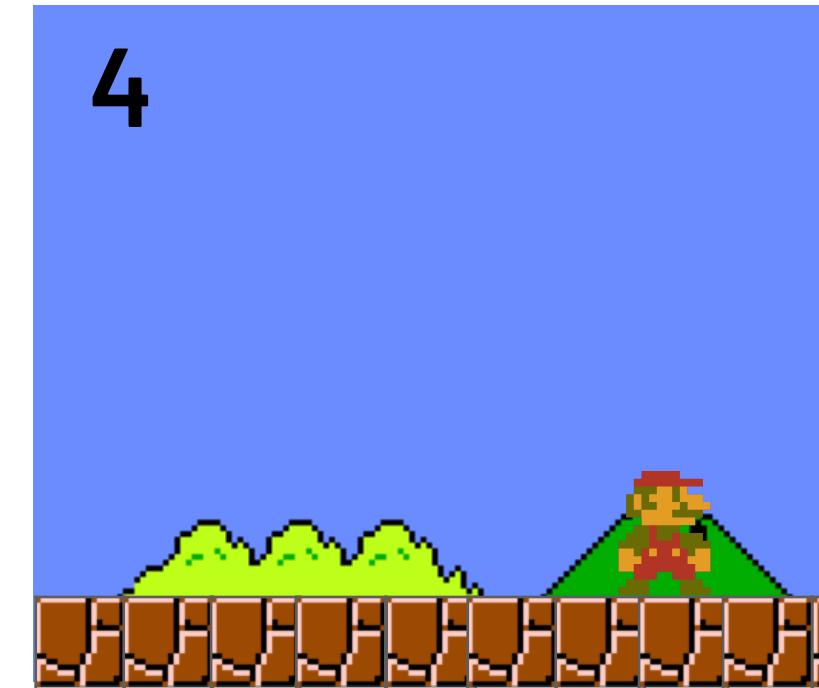
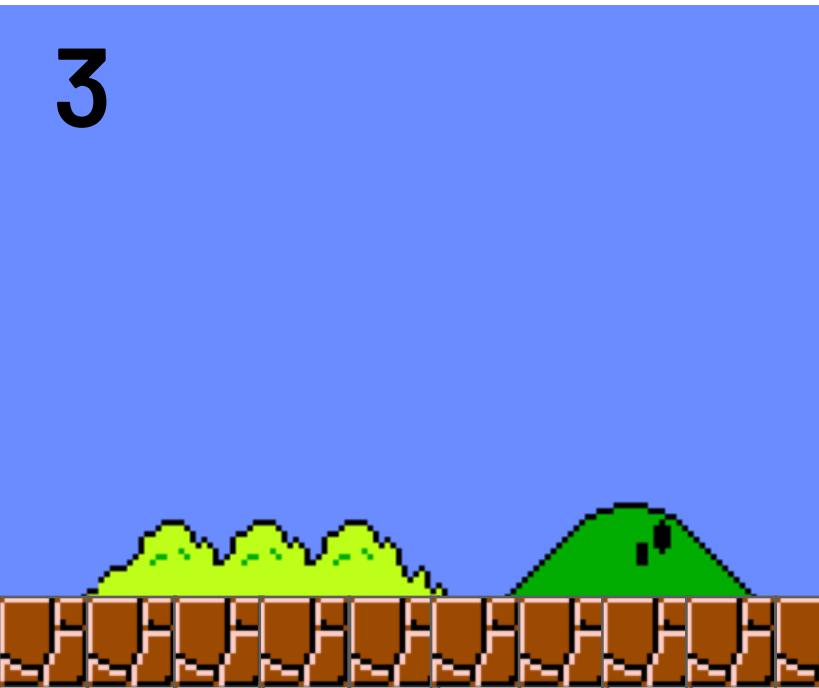
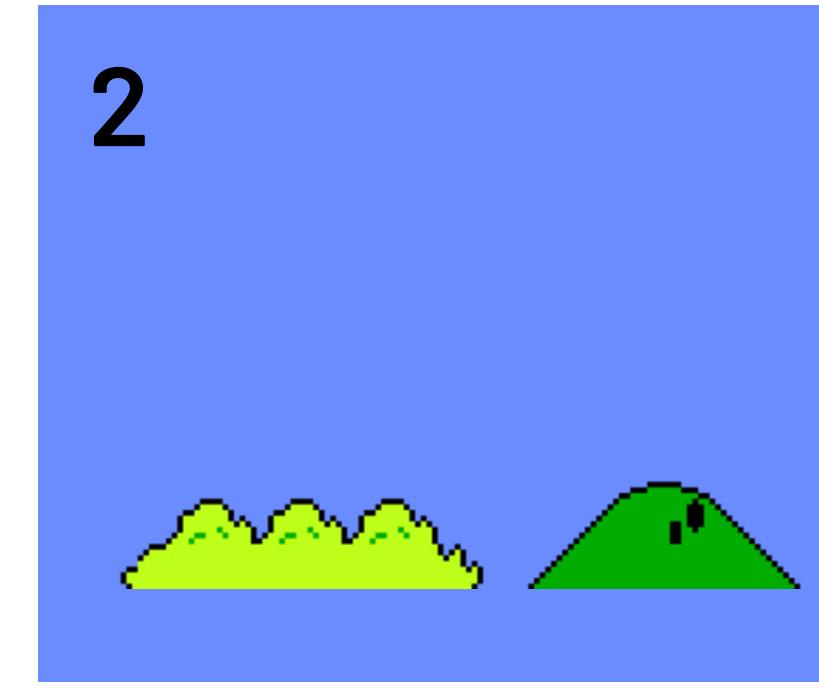
    // Inside the game loop
    SDL_RenderCopy(renderer, myTexture, nullptr, nullptr);

    ...
}
```

Desenhando Sprites



Sprites são normalmente desenhados seguindo o **algoritmo do pintor**: manter uma lista ordenada de sprites e desenhá-la de trás pra frente.



```
SortedList spriteList

// When creating a new sprite...
Sprite newSprite = specify image and desired x/y
newSprite.drawOrder = set desired draw order value

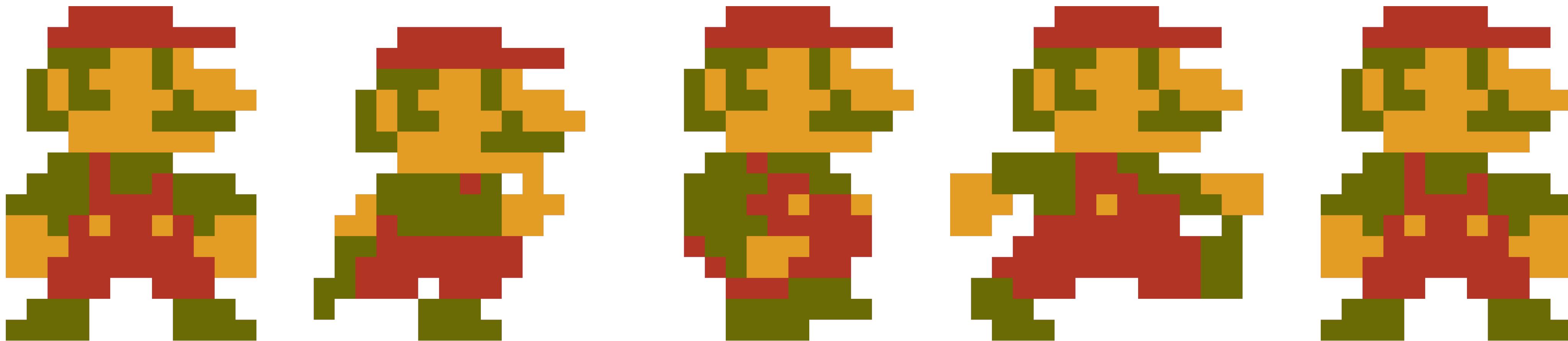
// Add to sorted list based on draw order value
spriteList.Add(newSprite.drawOrder, newSprite)

// When it's time to draw...
foreach Sprite s in spriteList
    s.Draw()
```

Animando Sprites

m

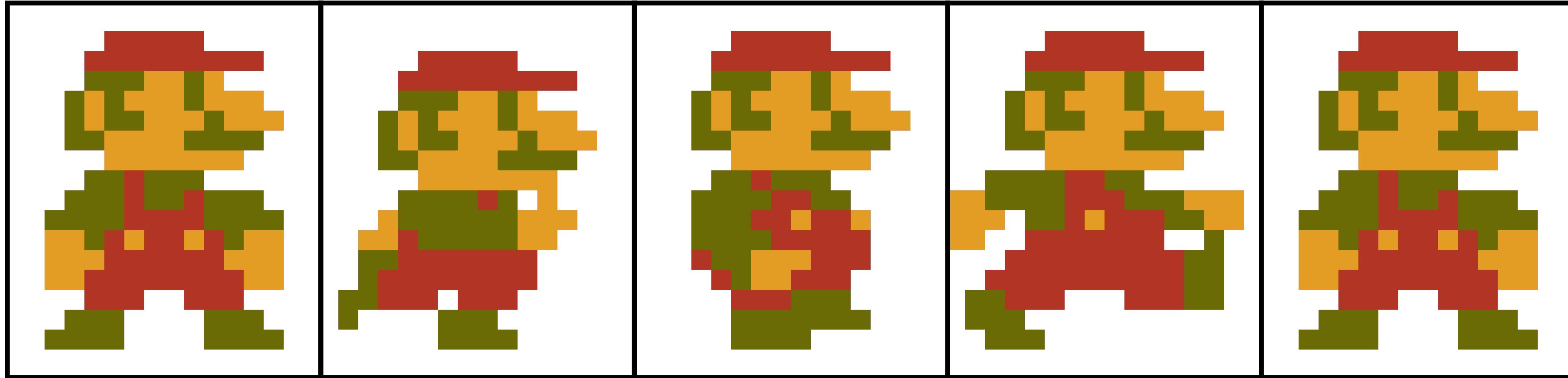
Para criar animações usando sprites, uma série de imagens estáticas reproduzidas em rápida sucessão para criar uma ilusão de movimento.



Armazenando Sprites

m

Armazenar sprites em arquivos separados pode acabar desperdiçando muita memória e processamento (considerando sprites com imagens de tamanhos iguais).



idle.png

run1.png

run2.png

run3.png

Sprite Sheets

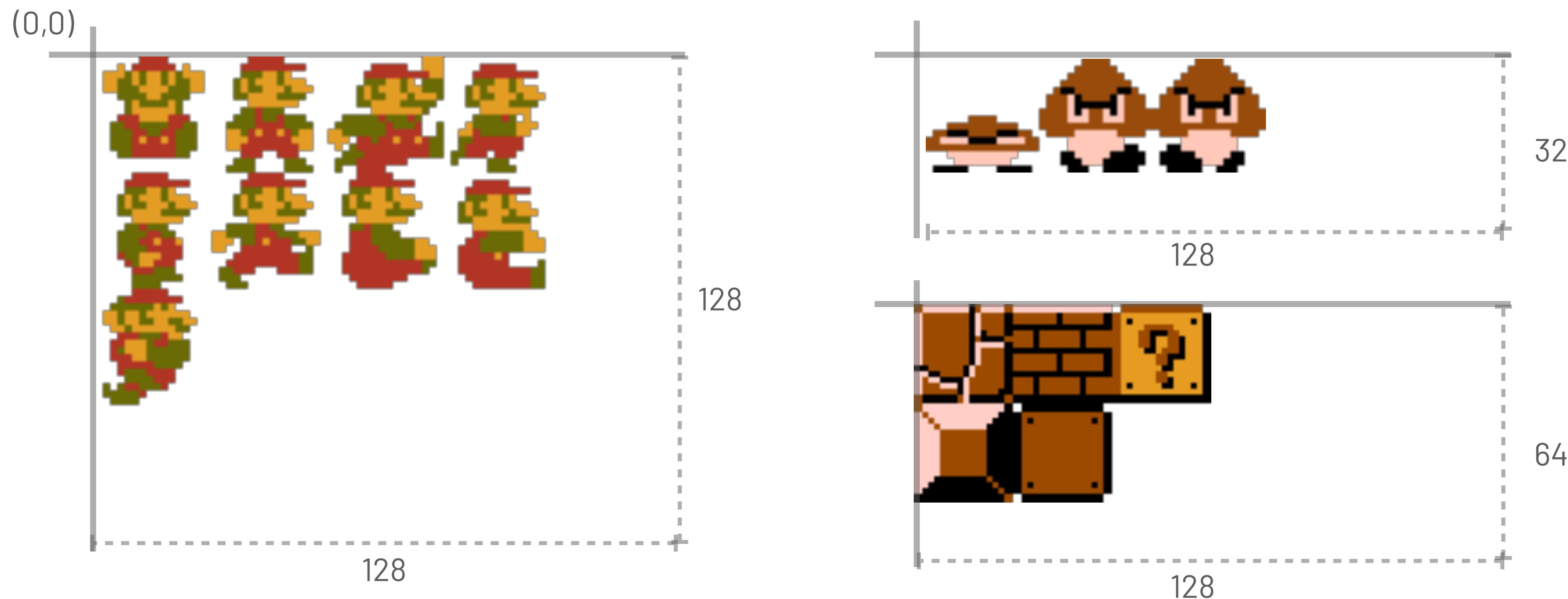
Para otimizar o espaço de armazenamento, geralmente agrupamos os sprites do jogo em um único imagem maior chamada de Sprite Sheet.



Historicamente, sprite sheets são criados em imagens com tamanho em potência de 2

Sprite Sheets

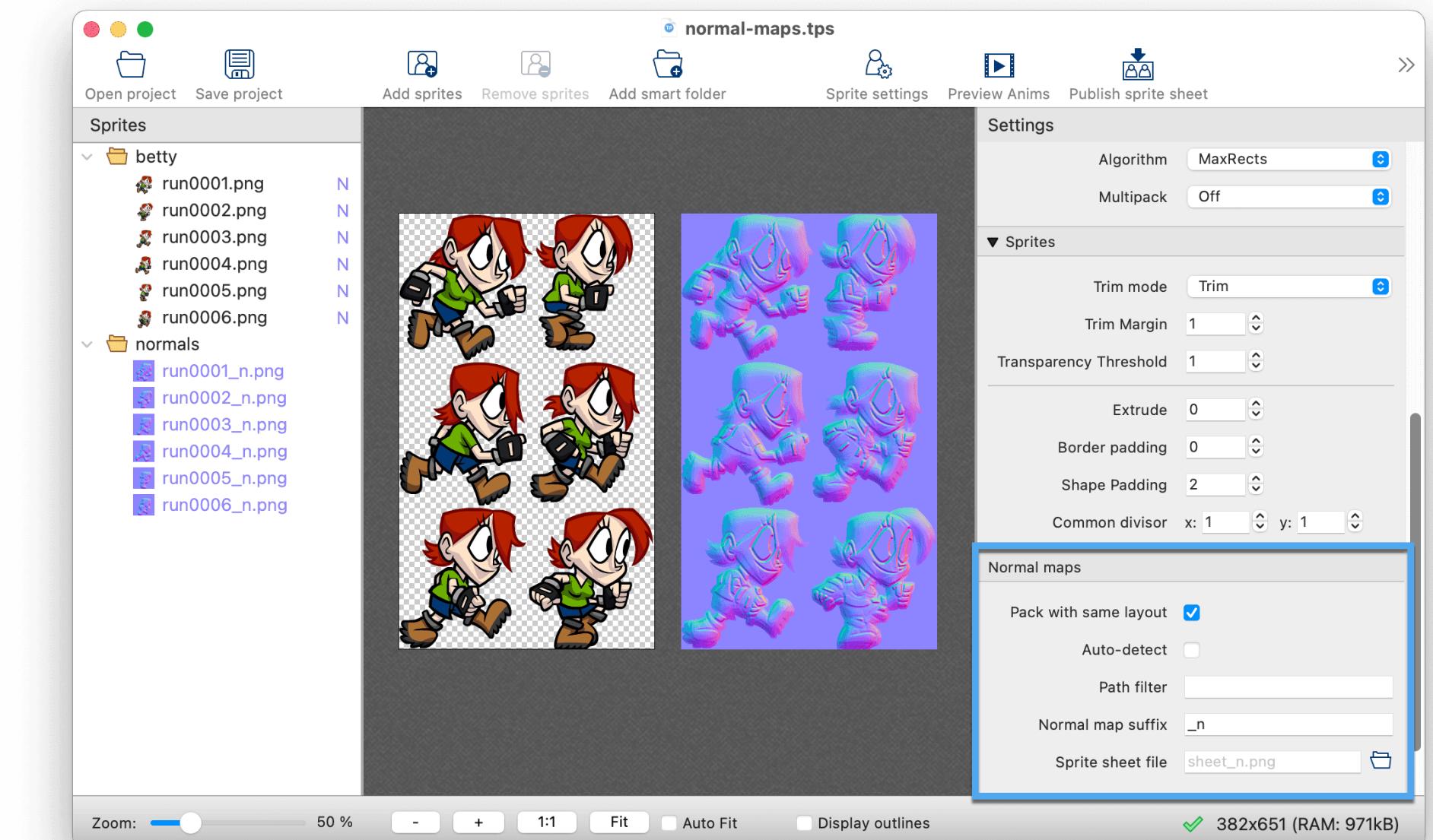
Opcionalmente, podemos criar um sprite sheet para cada personagem, facilitando a indexação de sprites e execução de animações.



Criando Sprite Sheets

Existem vários editores que auxiliam a criação de Sprite Sheets de maneira automática semi-automática:

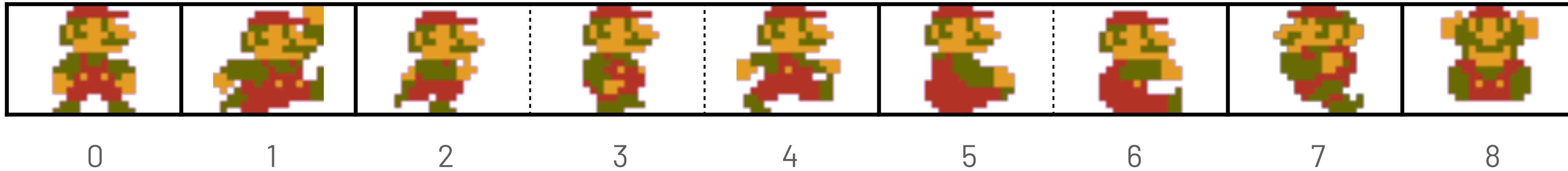
- ▶ Texture Packer
<https://www.codeandweb.com/texturepacker>
- ▶ Free Texture Packer
<https://free-tex-packer.com/>
- ▶ Piskel
<https://www.piskelapp.com/p/create/sprite>
- ▶ Aseprite
<https://www.aseprite.org/>



Representando Animações



Manter uma lista de imagens com todos os quadros de um personagem:



Manter uma lista com os indices dos quadros de cada animação do personagem:

Idle	[0]
Jump	[1]
Run	[2, 3, 4]
Stomp	[5, 6]
Turn	[7]
Dead	[8]

Tocando Animações



Não podemos assumir que a taxa de quadros da animação seja mais lenta que a taxa de quadros do jogo

- ▶ FPS do Jogo: 30
- ▶ FPS de uma animação com 24 quadros: 48

Isso significa que muitas vezes precisaremos pular vários quadros na animação.

Tocando Animações



Precisamos de dois floats para tocar uma animação:

- ▶ AnimTimer, para armazenar o tempo corrente da animação
- ▶ AnimFPS, para armazenar a taxa de atualização da animação

Transformar (cast) **AnimTimer** para inteiro para acessar o índice da animação

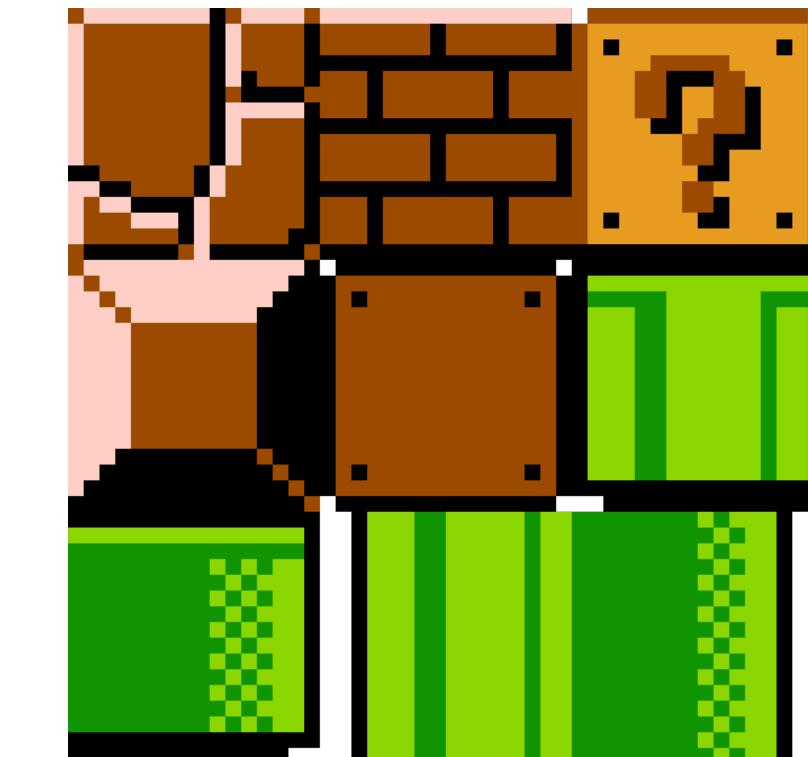
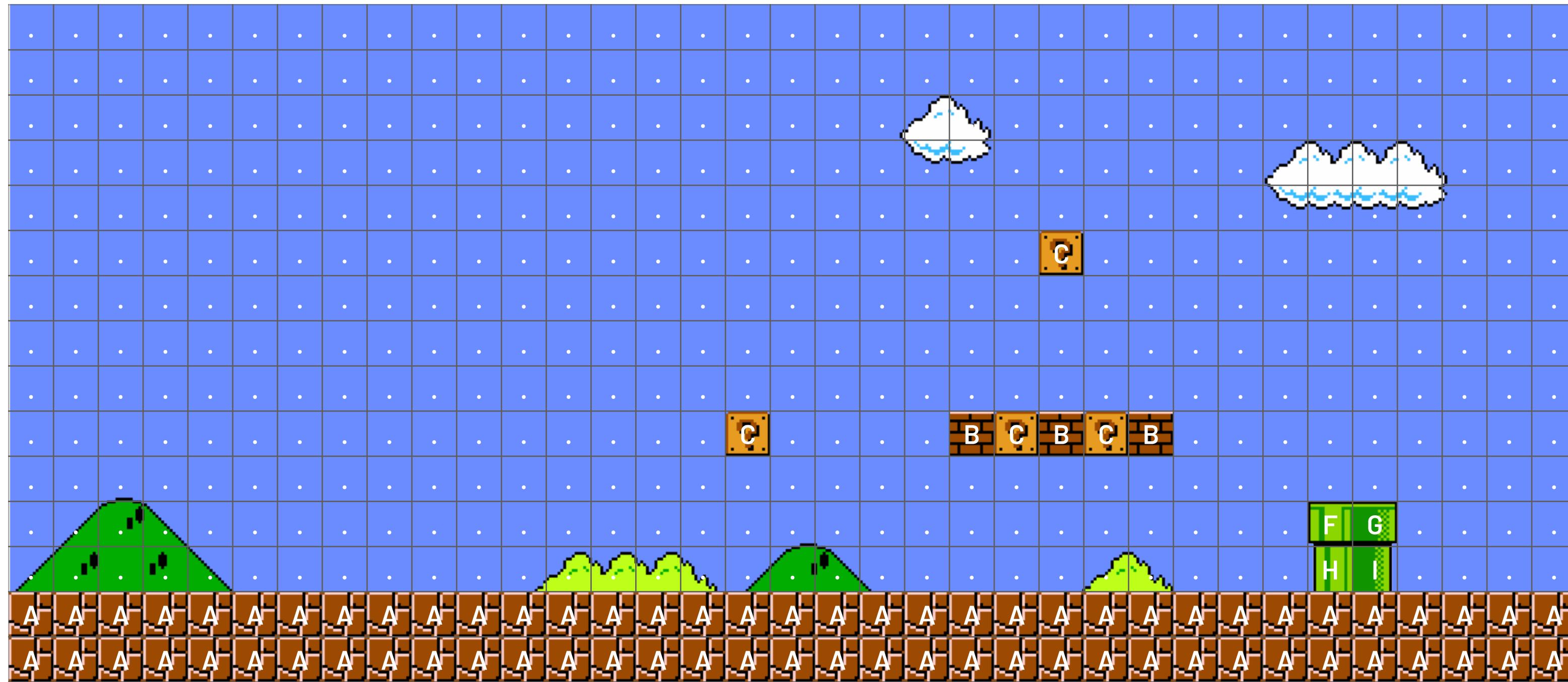
```
frameTime += deltaTime

// verificar se está na hora de alterar o sprite
if frameTime > (1 / animFPS) {
    animTimer = frameTime * animFPS // frameTime / (1 / animFPS) -> frameTime * animFPS
    if animTimer >= animData.frameInfo[animNum].numFrames:
        animTimer = (int)animTimer % animData.frameInfo[animNum].numFrames
}

int imageNum = animData.frameInfo[animNum].startFrame + frameNum
```

Tilemaps

Tilemaps são uma forma de organizar o mundo do jogo em uma grade de células de tamanhos iguais, cada um com número identificador, visando maximizar a repetição de sprites.

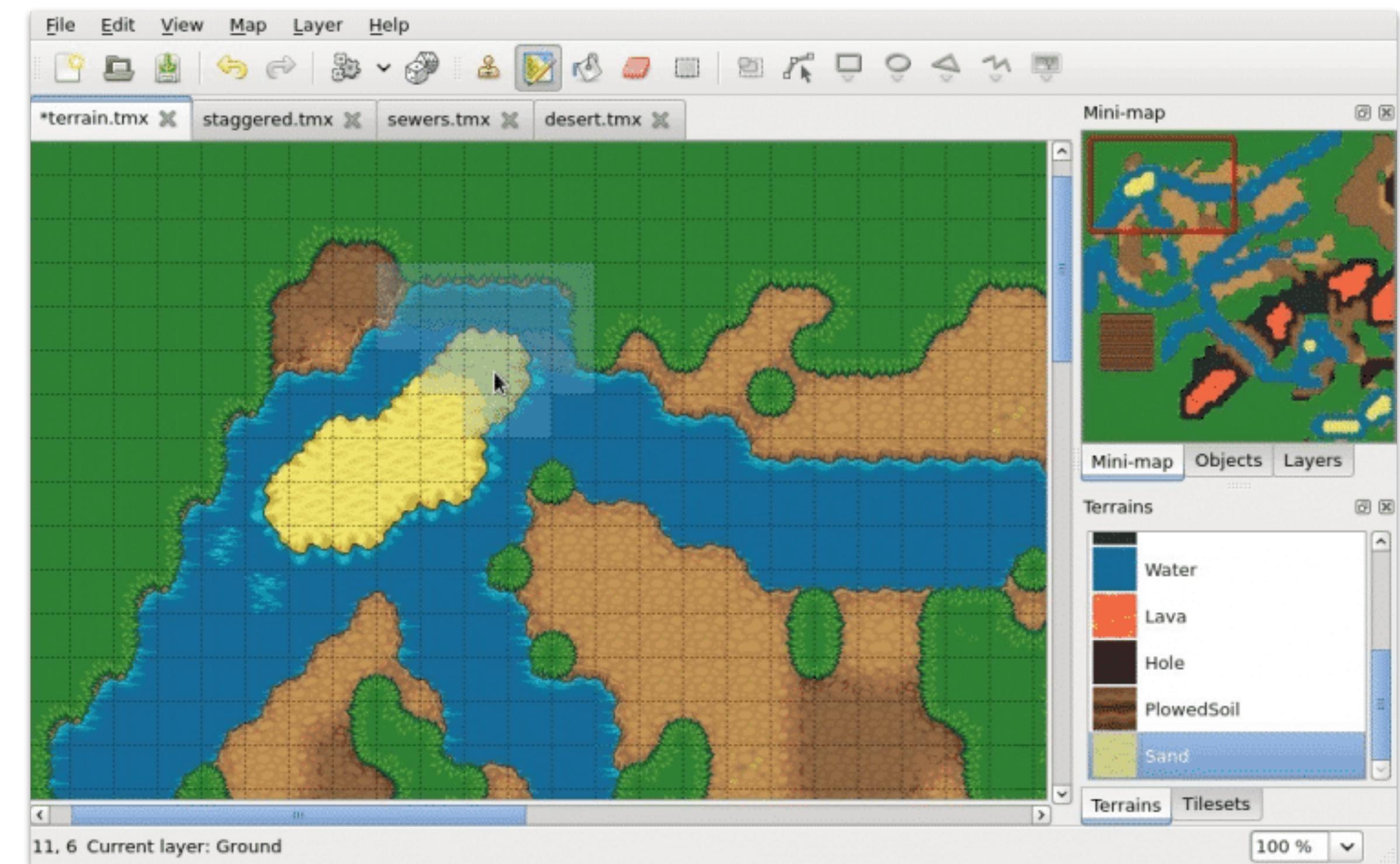


A estrutura de grade (grid) dos tilemaps facilita a edição de níveis, pois a posição dos tiles é limitado a coordenadas discretas.

Criando Tilemaps

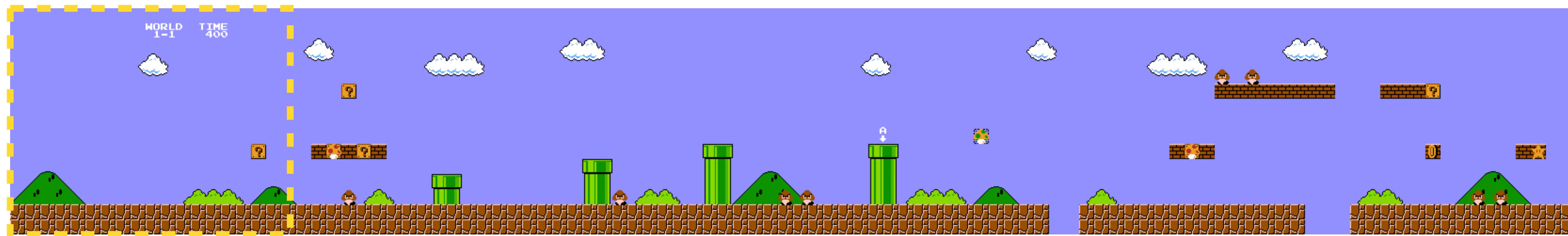
Existem vários editores que auxiliam a criação de **Tilemaps**. Além de facilitar a criação dos mapas em si, eles geralmente possibilitam a definição de colisões estáticas, triggers, entre outros:

- ▶ Tiled
<https://www.mapeditor.org/>
- ▶ Sprite Fusion
<https://www.spritefusion.com/>
- ▶ PixLab 2D Tilemap Maker
<https://tilemap.pixlab.io/>



Rolagem de Câmera

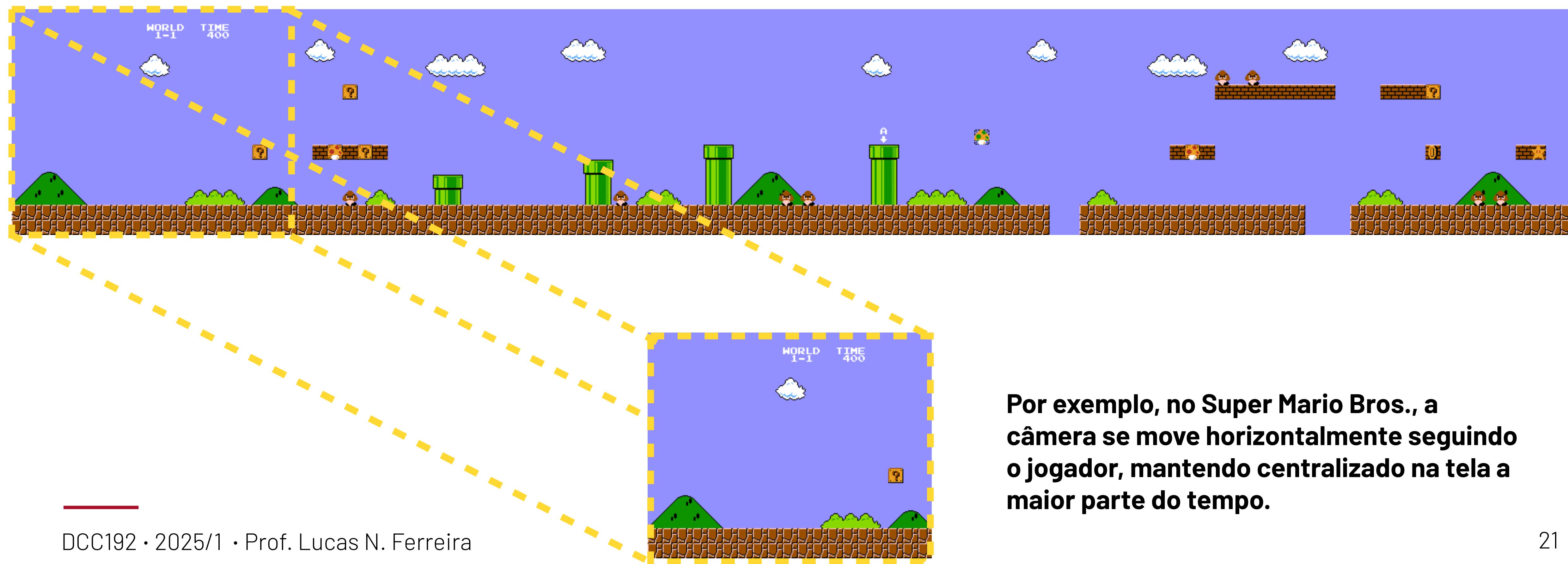
É muito comum que jogos 2D tenham mundos grandes que não cabem na tela. Nesse caso, precisamos implementar uma câmera, que se move para mostrar a região de interesse atual.



Por exemplo, no Super Mario Bros., a câmera se move horizontalmente seguindo o jogador, mantendo centralizado na tela a maior parte do tempo.

Rolagem de Câmera

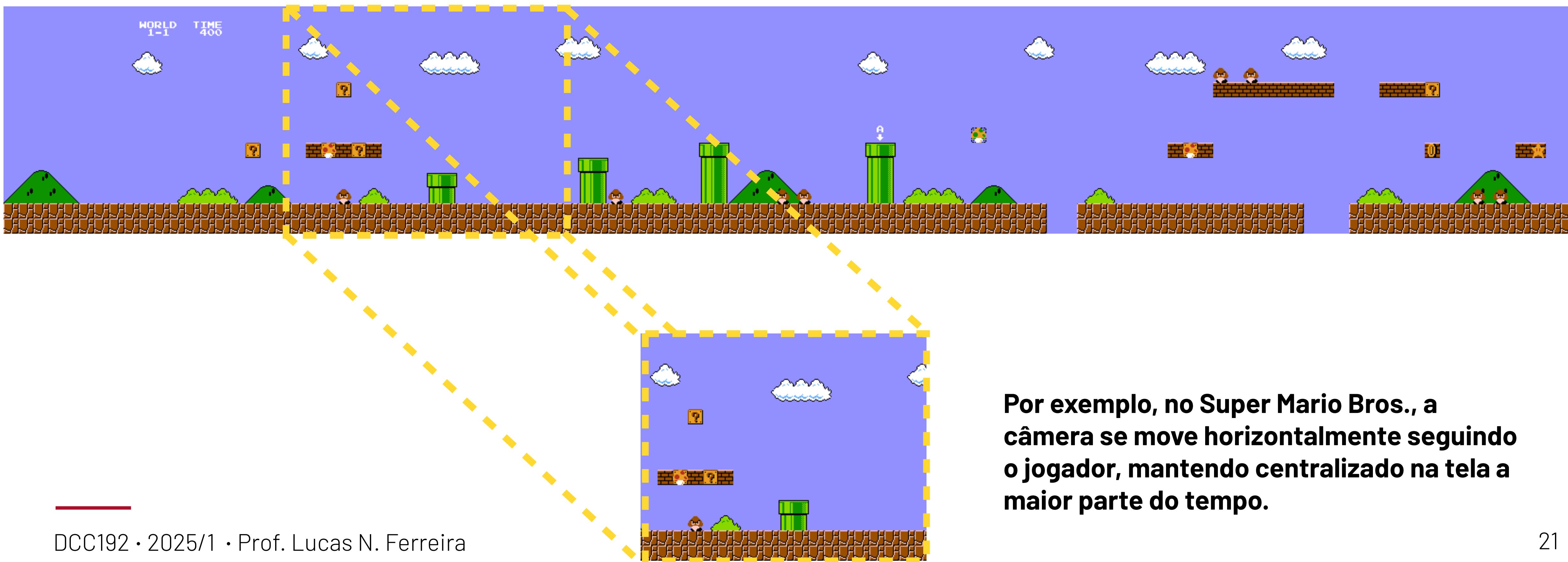
É muito comum que jogos 2D tenham mundos grandes que não cabem na tela. Nesse caso, precisamos implementar uma câmera, que se move para mostrar a região de interesse atual.



Por exemplo, no Super Mario Bros., a câmera se move horizontalmente seguindo o jogador, mantendo centralizado na tela a maior parte do tempo.

Rolagem de Câmera

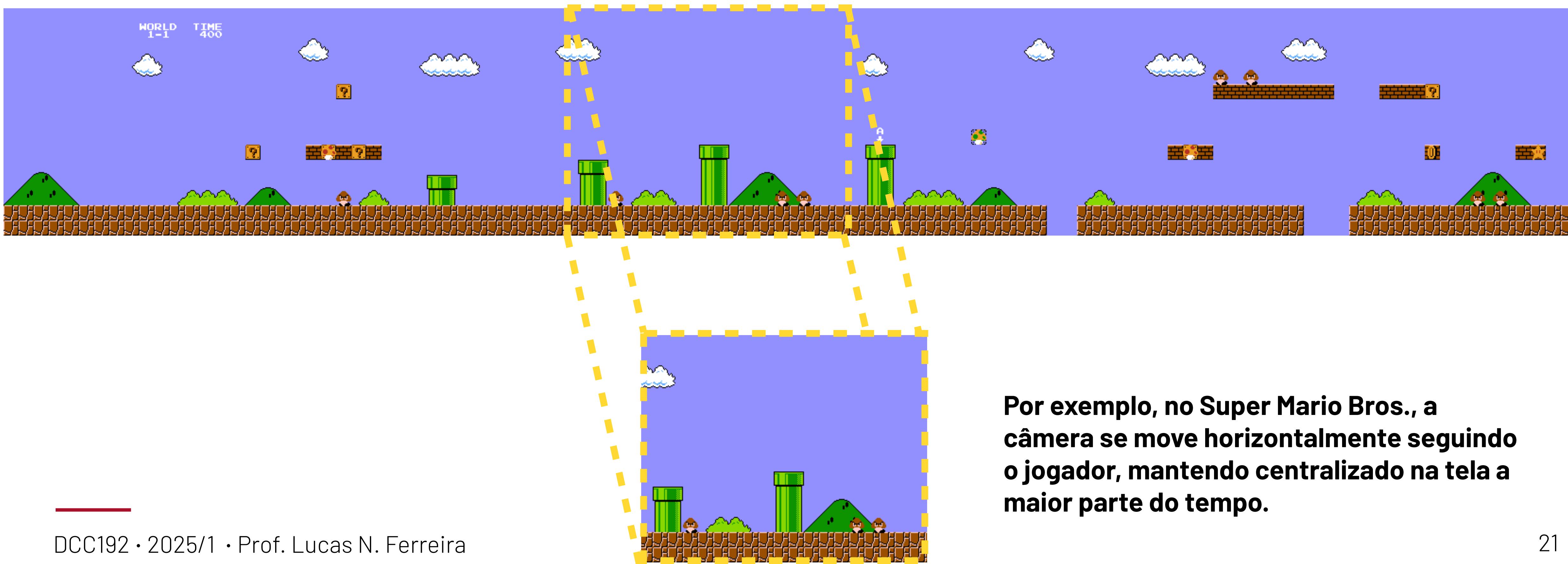
É muito comum que jogos 2D tenham mundos grandes que não cabem na tela. Nesse caso, precisamos implementar uma câmera, que se move para mostrar a região de interesse atual.



Por exemplo, no Super Mario Bros., a câmera se move horizontalmente seguindo o jogador, mantendo centralizado na tela a maior parte do tempo.

Rolagem de Câmera

É muito comum que jogos 2D tenham mundos grandes que não cabem na tela. Nesse caso, precisamos implementar uma câmera, que se move para mostrar a região de interesse atual.

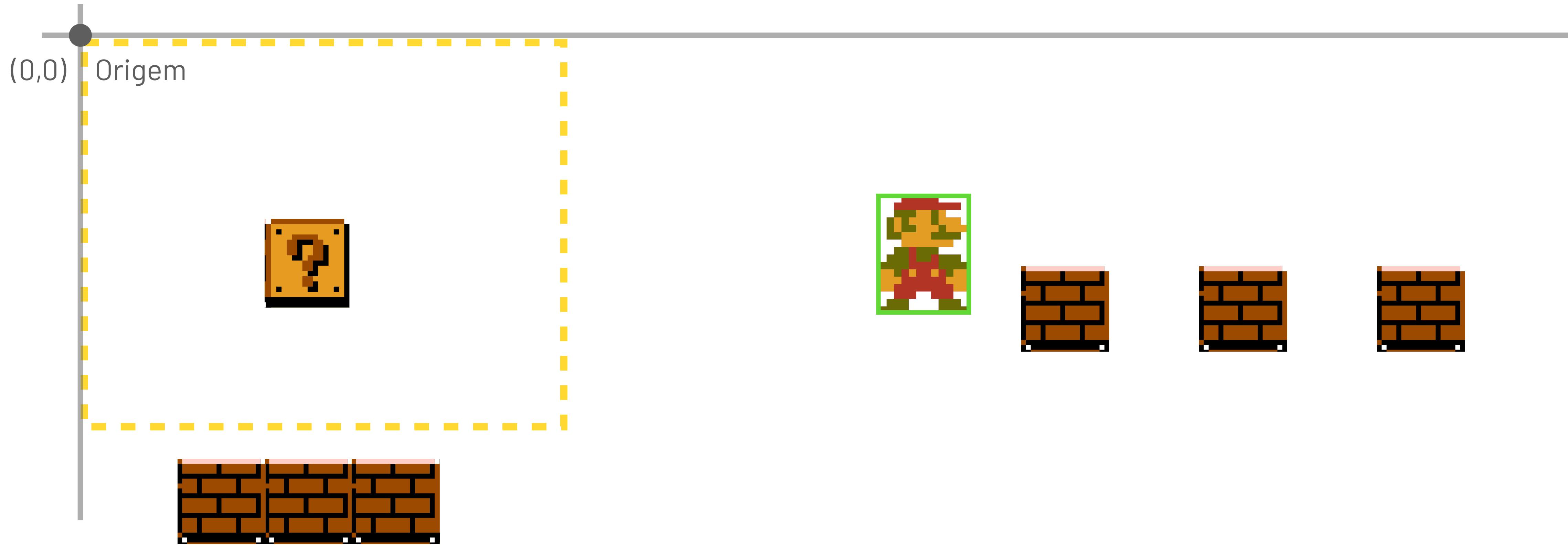


Por exemplo, no Super Mario Bros., a câmera se move horizontalmente seguindo o jogador, mantendo centralizado na tela a maior parte do tempo.

Rolagem de Câmera

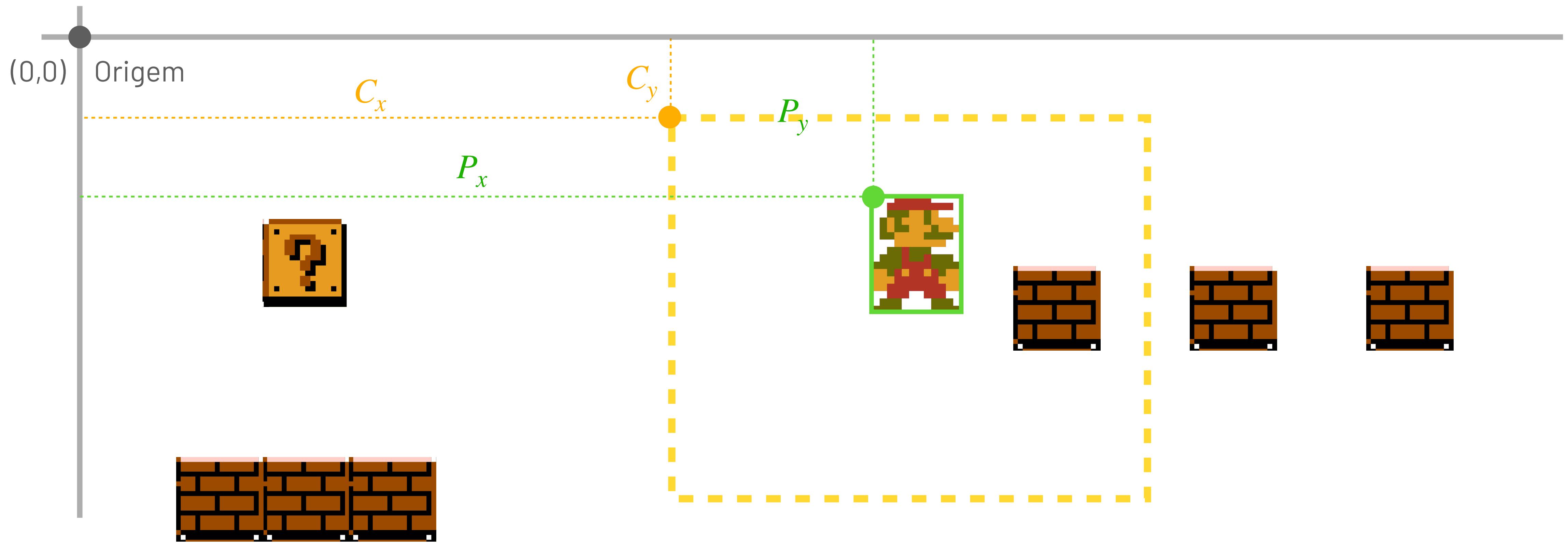


Originalmente, os objetos são desenhados com relação à origem do mundo (o canto superior esquerdo da tela). Porém, objetos que estão fora da tela não aparecem!



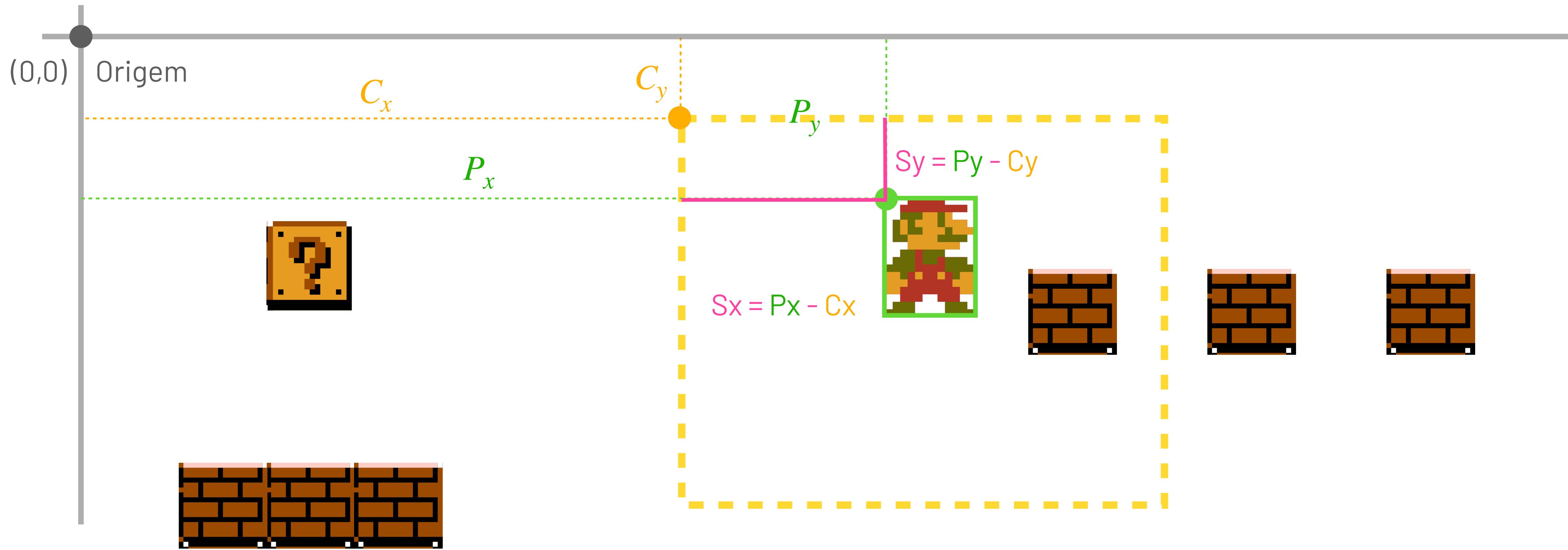
Rolagem de Câmera

Basta desenhar os objetos com relação à posição da **câmera C** , representada por uma posição relativa à origem do mundo.



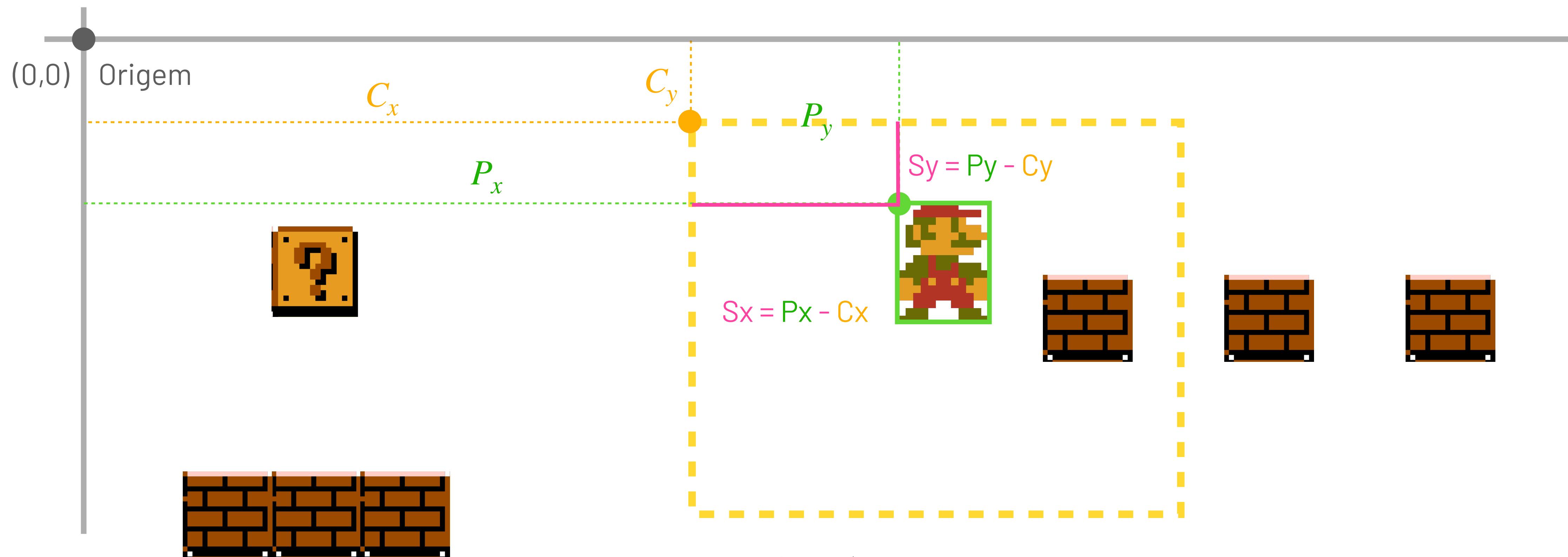
Rolagem de Câmera

Basta desenhar os objetos com relação à posição da **câmera C** , representada por uma posição relativa à origem do mundo.



Rolagem de Câmera

Basta desenhar os objetos com relação à posição da **câmera C** , representada por uma posição relativa à origem do mundo.



$$\vec{S} = \vec{P} - \vec{C}$$

screenPosition = worldPosition - cameraPosition

Efeito de Paralaxe

Objetos mais distantes se movem mais lentamente do que objetos mais próximos:

Para implementar esse efeito, basta multiplicar a posição da câmera \vec{C} por um fator de paralaxe p :

$$\vec{S} = \vec{P} - p \vec{C}$$

Por exemplo:

- $p = 1.0$ (camada do jogador)
- $p = 0.5$ (camada do meio)
- $p = 0.25$ (camada do fundo)



Próxima aula



A11: Gráficos 2D

- ▶ Design de Câmeras 2D
- ▶ Menus
- ▶ Heads-up Display