DCC192

2025/2

UFMG

Desenvolvimento de Jogos Digitais

A11: Gráficos 2D

Prof. Lucas N. Ferreira

Logística

Avisos

- O **TP2: Asteroids** será disponibilizado até amanhã!

Última Aula

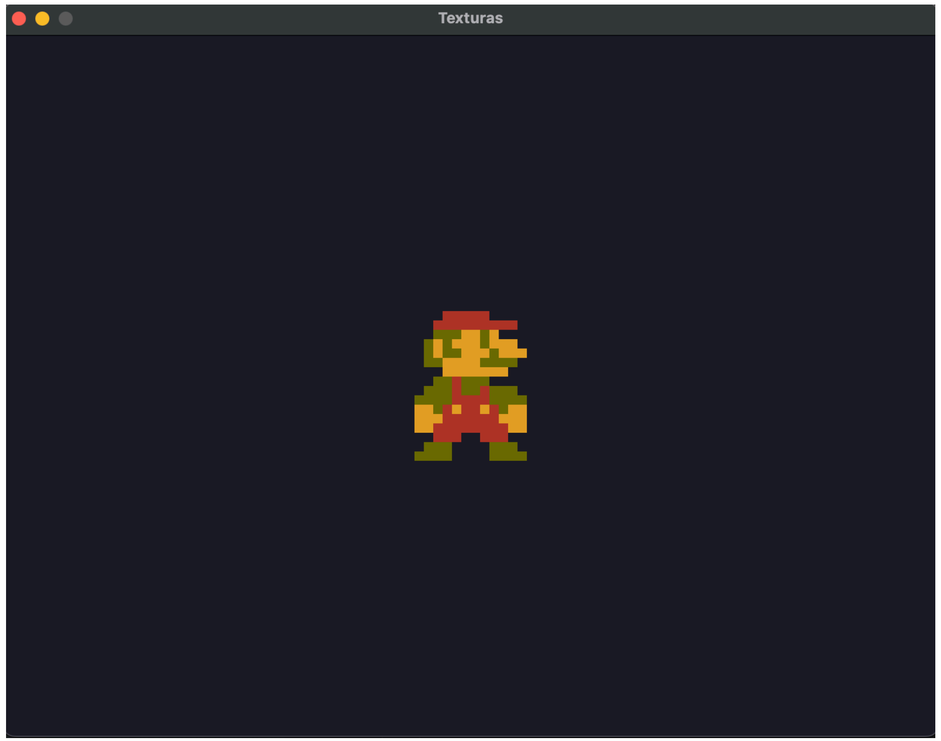
- Sistemas de Partículas

Plano de Aula

- Sprites e Spritesheets
- Mapeamento de Texturas em OpenGL
  - Coordenadas uv
- Animações 2D
- Desenhando Mapas
  - Tileset
  - Composição Manual

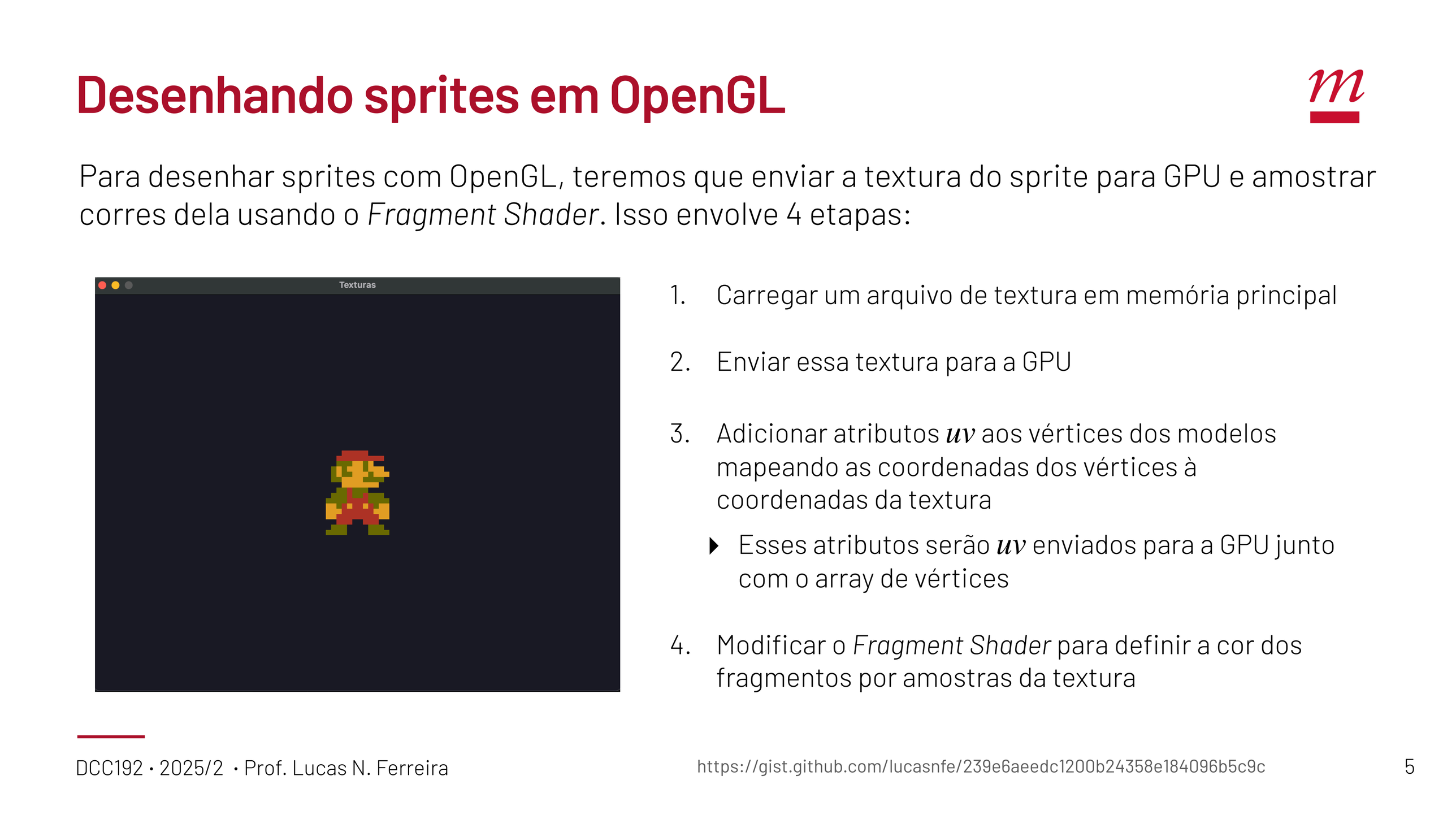
Sprites

Um **sprite** é uma imagem utilizada para representar um estado (ou pose) de um objeto do jogo visualmente em 2D. Um sprite é definido por:



- *Textura*: arranjo bidimensional com os pixels que formam o sprite
  - Por exemplo: mario_idle.png
- *Posição*: local de desenho na tela
  - Por exemplo: (100,100)
- *Ordem de desenho*: define qual sprite é desenhado primeiro
  - Por exemplo: 99

Desenhando sprites em OpenGL



Para desenhar sprites com OpenGL, teremos que enviar a textura do sprite para GPU e amostrar corres dela usando o *Fragment Shader*. Isso envolve 4 etapas:

1. Carregar um arquivo de textura em memória principal
2. Enviar essa textura para a GPU
3. Adicionar atributos uv aos vértices dos modelos mapeando as coordenadas dos vértices à coordenadas da textura
   - Esses atributos serão uv enviados para a GPU junto com o array de vértices
4. Modificar o *Fragment Shader* para definir a cor dos fragmentos por amostras da textura

https://gist.github.com/lucasnfe/239e6aeedc1200b24358e184096b5c9c

Carregando Texturas com SDL

A SDL possui uma bilioteca auxiliar, `SDL_Image.h`, para carregar e manipular texturas:

1. Inicializar a SDL_Image:

```
if (IMG_Init(IMG_INIT_PNG) == 0) {
    SDL_Log("Unable to initialize SDL_image: %s", SDL_GetError());
    return false;
}
```

2. Usar a função IMG_Load para carregar uma textura do disco:

```
SDL_Surface* surf = IMG_Load(“mario.png”);
if (!surf) {
    SDL_Log("Failed to load texture file %s", texturePath.c_str());
    return nullptr;
}
```

Enviar Textura para a GPU

Com a textura carregada em memória `s`, podemos enviá-la para a GPU em duas etapas :

1. Alocar um bloco memória de textura na GPU e transferir os dados da RAM para lá:

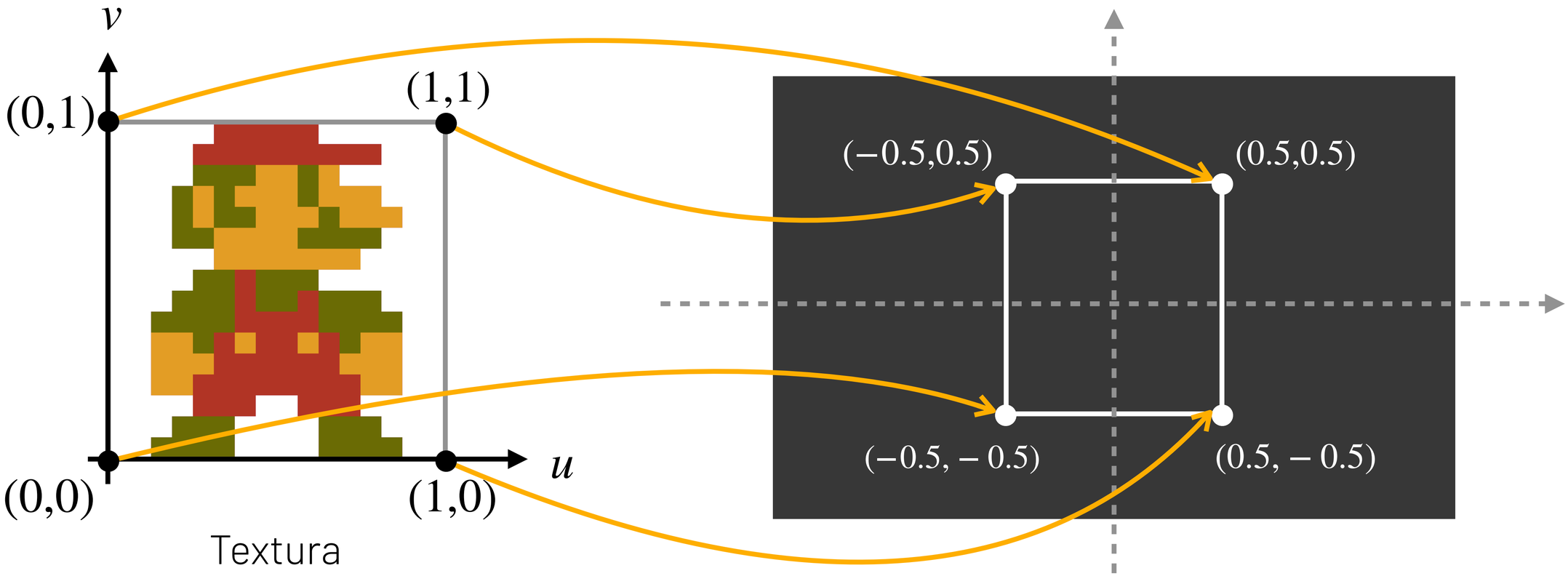
```
GLuint textureID;
glGenTextures(1, &textureID); // Cria textura na GPU
glBindTexture(GL_TEXTURE_2D, textureID); // Linka a textura para configuração
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, s->w, s->h, 0, GL_BGRA, GL_UNSIGNED_BYTE, s->pixels);
```

2. Configura parâmetros da textura (ex., filtro de maximização e minimização)

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

Mapeando Vértices às Coordenadas de Texturas

Em OpenGL, as texturas possuem um espaço de coordenadas normalizado (u, v) e, para cada vértice do modelo, temos que definir qual coordenada (u, v) será mapeado para aquele vértice:



Amostrando Cores de Texturas

Com a textura carregada e ativa em GPU, podemos criar uma variável uniform no fragment shader para amostrar cores dessa textura:

1. A função `texture()` da GLSL nos permite amostrar uma cor de uma textura passando uma coord.:

```
in vec2 TexCoord; // Coordenada de textura do vértice, passada via vertex shader
out vec4 outColor;
uniform sampler2D uTexture;
void main() {
    outColor = texture(uTexture, TexCoord);
}
```

2. Alterar valor da variável uTexture para 0, que é a unidade que carregamos a nossa textura:

```
GLint uTexture = glGetUniformLocation(shaderProgram, "uTexture");
glUniform1i(uTexture, 0); // Altera valor da variável para a região de textura 0
```

Amostrando Cores de Texturas

Com a textura carregada e ativa em GPU, podemos criar uma variável uniform no fragment shader para amostrar cores dessa textura:

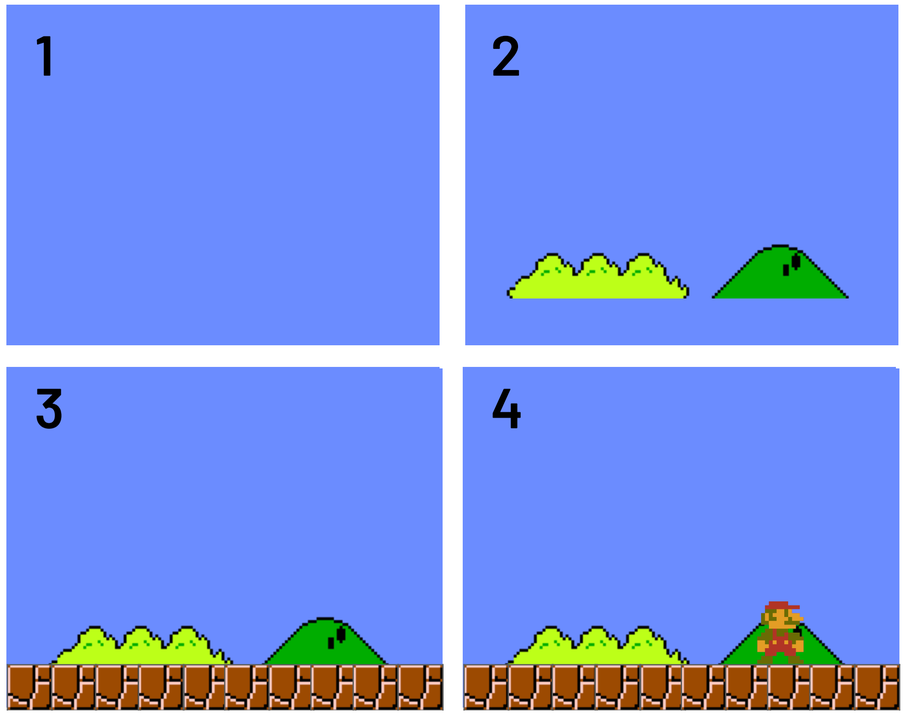
3. Por último, temos que ativar a unidade de textura desejada e linkar a textura no pipeline gráfico pelo seu ID:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID);
```

Se tivermos mais de uma textura carregada (ex. Múltiplos sprite sheets), essas chamadas de ativação e link deverão ser feitas antes de cada chamada `glDrawElements` para ativar a textura desejada.

Desenhando Sprites

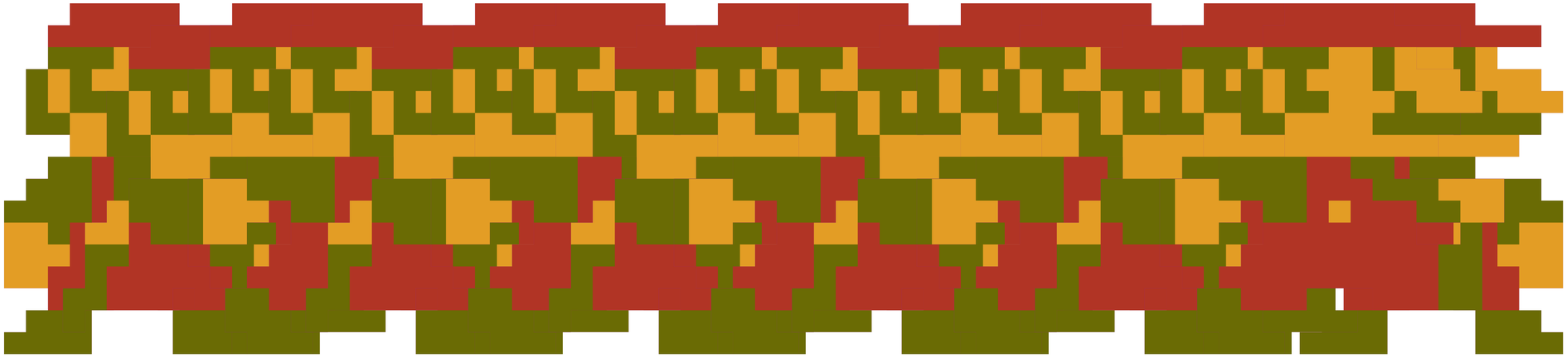
Sprites são normalmente desenhados seguindo o **algoritmo do pintor**: manter uma lista ordenada de sprites e desenhá-la de trás pra frente.




```
SortedList spriteList;

// When creating a new sprite...
Sprite newSprite = new Sprite(“img.png”);
newSprite->SetPosition(Vector2(10, 10));
newSprite->SetDrawOrder();

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

// When it's time to draw...
for (Sprite *s : spriteList)
   s->Draw()
```

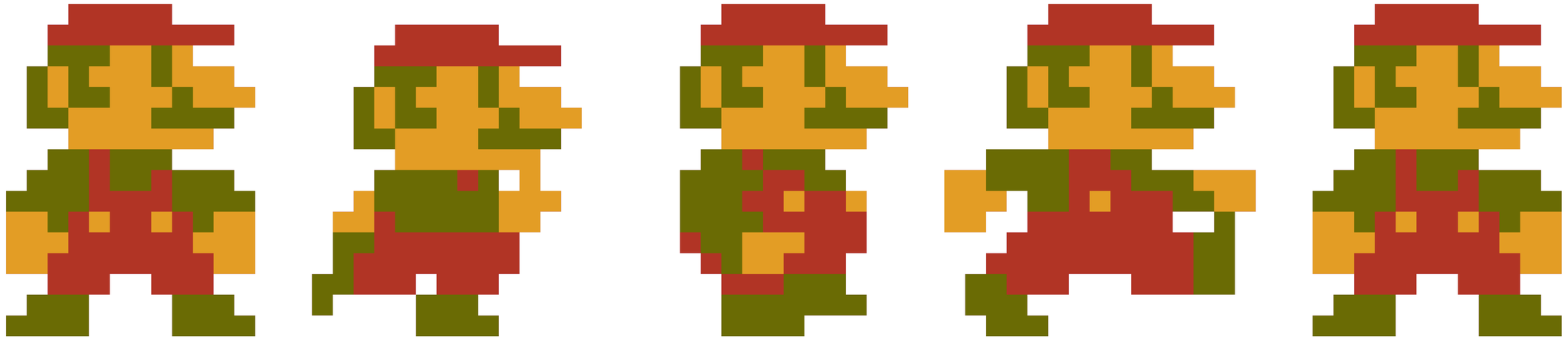
Animando Sprites

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



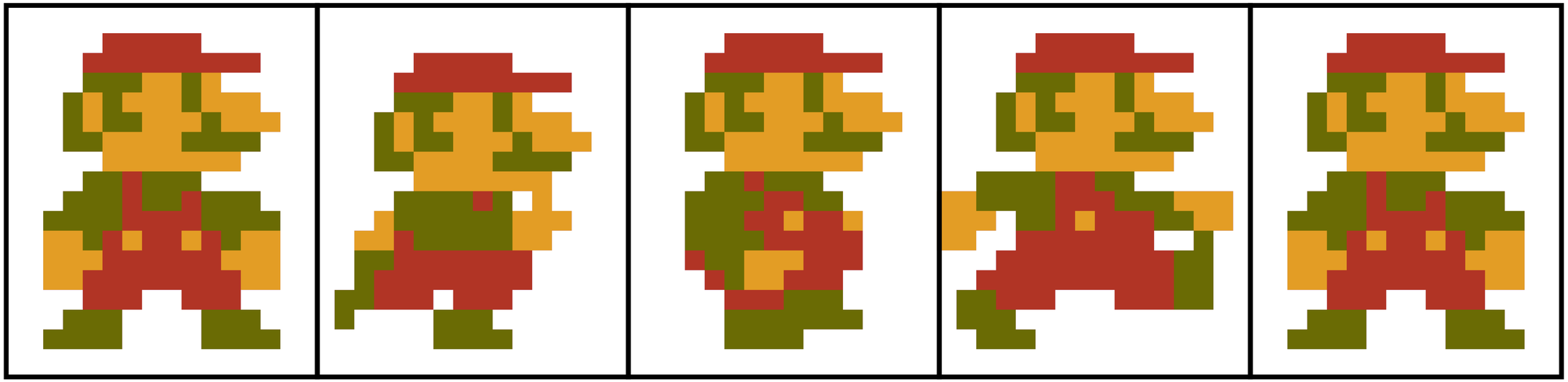
Animando Sprites

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



Armazenando Sprites

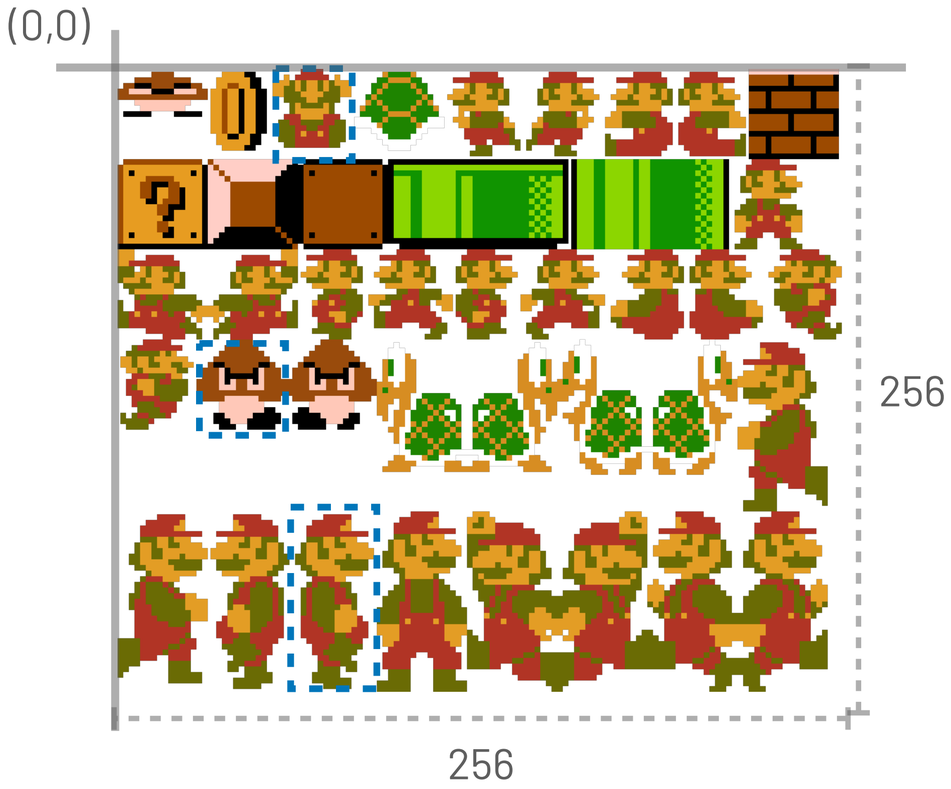
Armazenar sprites em arquivos separados pode acabar desperdiçando muita memória e processamento (considerando sprites com imagens de tamanhos iguais).



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.



- A imagem do sprite sheet deve ser acompanhada de um arquivo auxiliar (ex. json) que lista a posição e tamanho de cada sprite:

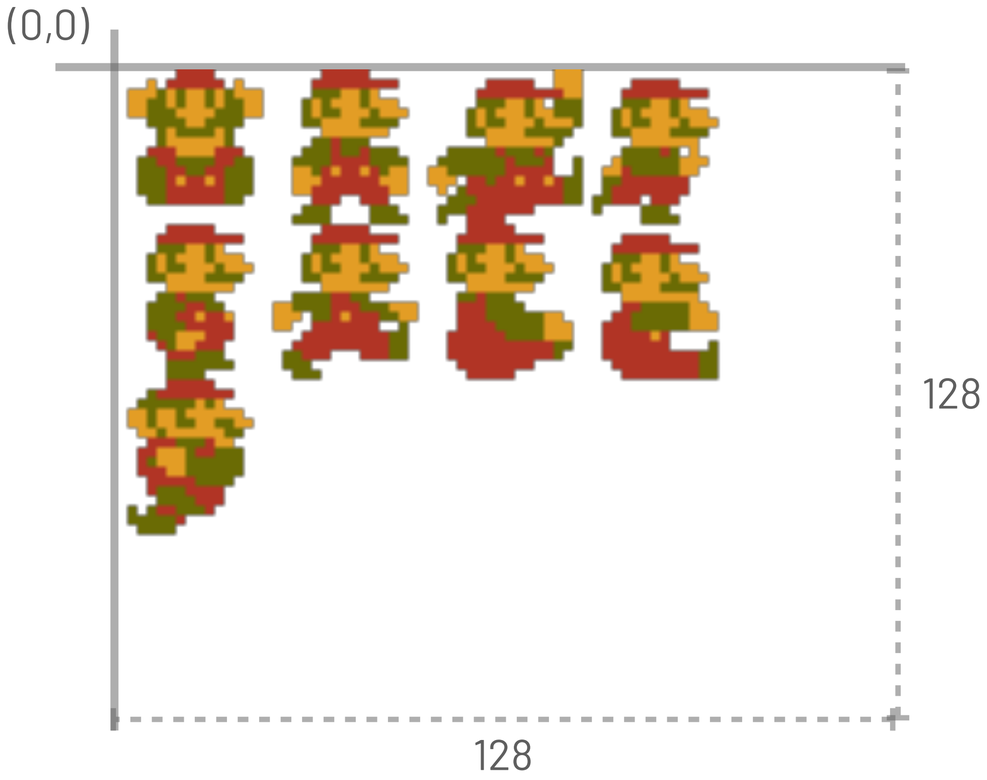

{x: 64, y: 0, w: 32, h: 32}

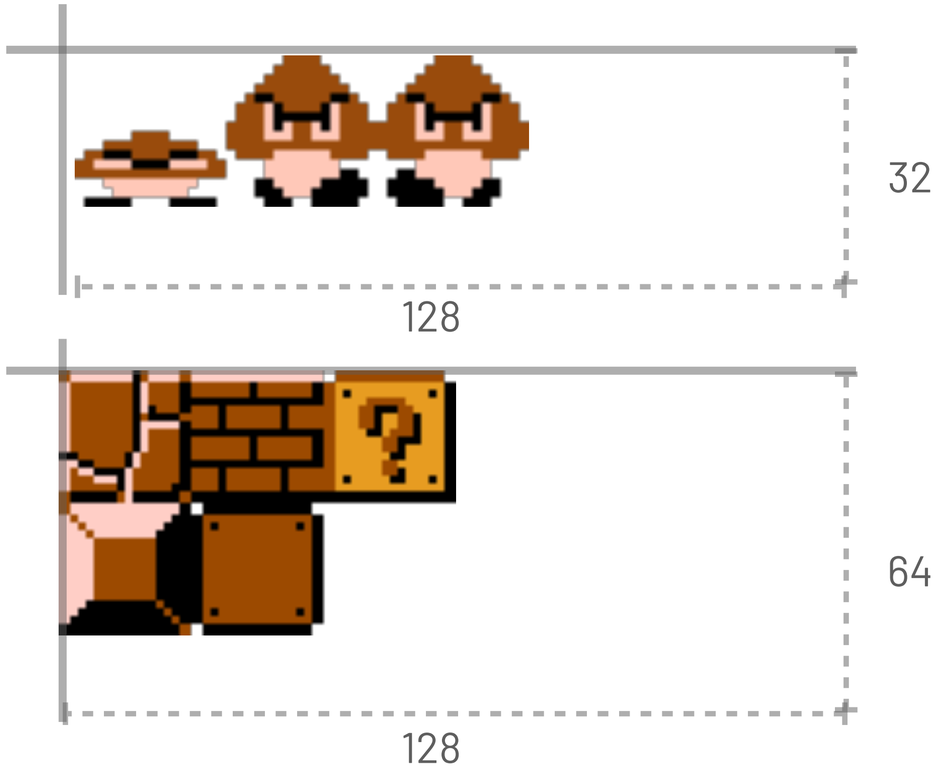

{x: 32, y: 128, w: 32, h: 32}

Historicamente, sprite sheets são criados em imagens com tamanho em potência de 2, por restrições das GPUs

Sprite Sheets

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



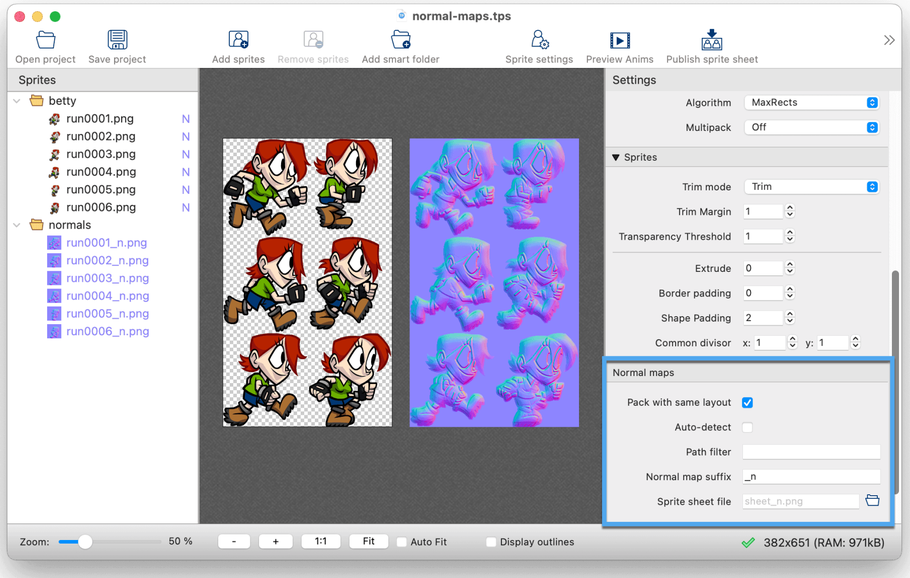


Dividir os sprites em múltiplos sprite sheets pode ser necessário dependendo da quantidade de sprites

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 um mapa com os indices dos quadros de cada animação do personagem:

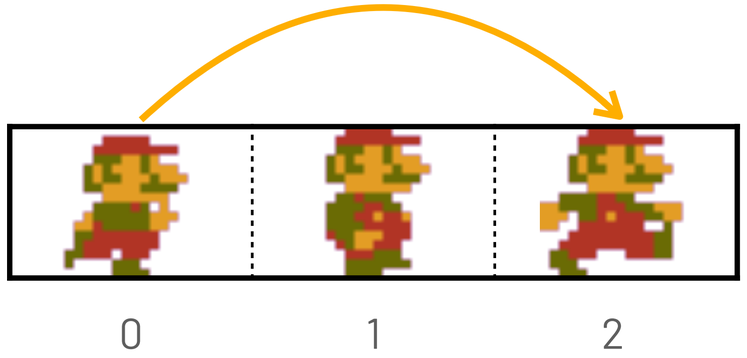
“Idle"	[0]
“Jump"	[1]
“Run”	[2, 3, 4]
“Stomp”	[5, 6]
“Turn”	[7]
“Dead"	[8]

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 quadros na animação:



Tocando Animações

Precisamos de dois *floats* para tocar uma animação:

- `mCurrFrame`, para armazenar o tempo corrente da animação
- `mAnimFPS`, para amazenar a taxa de atualização da animação

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

```
// Atualiza quadro corrente em função do FPS da animação
mCurrFrame += mAnimFPS * deltaTime;

// Manter o quadro corrente dentro dos limites da animação
while (mCurrFrame >= mAnimTextures[mAnimName].size()) {
  mCurrFrame -= mAnimTextures.size();
}

SetTexture(mAnimTextures[static_cast<int>(mCurrFrame)]);
```

0 1 2 3 4 5 6 7 8

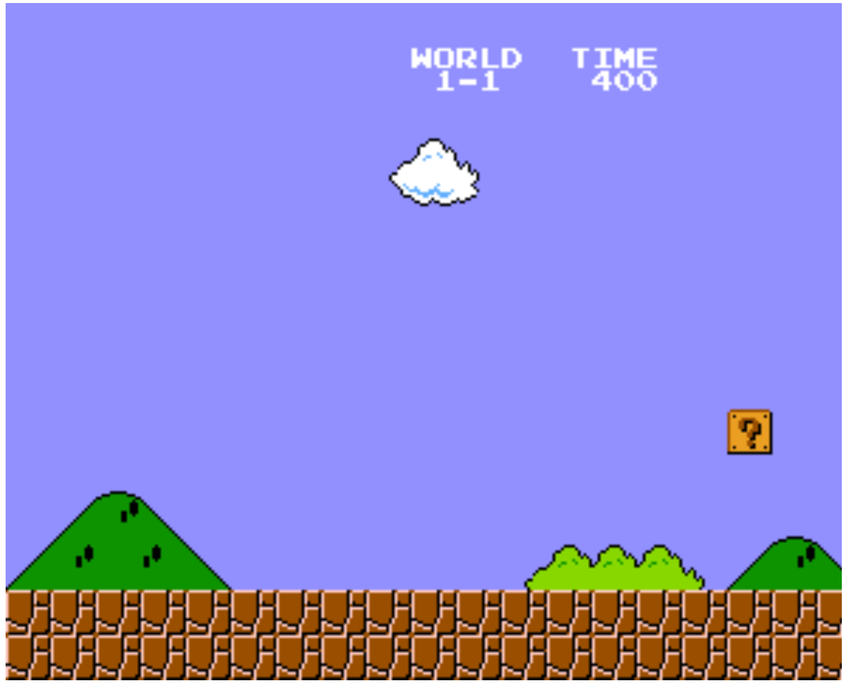
`mAnimTextures`

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

`mAnimName: ”Run”`

Desenhando Mapas

Até agora, vimos como desenhar os objetos dinâmicos por meio de animações via spritesheets, mas e o mapa do jogo, ou seja, os objetos estáticos que definem o ambiente do jogo?



Tilemaps

Por uma questão de otimização, Jogos 2D antigos, organizavam o mapa em uma estrutura de grade com células de mesmo tamanho.

Essas células podem ter formatos diferentes:

- **Quadrados**
- Hexagonais
- Isométricos

Desenhando Mapas

Até agora, vimos como desenhar os objetos dinâmicos por meio de animações via spritesheets, mas e o mapa do jogo, ou seja, os objetos estáticos que definem o ambiente do jogo?

Tilemaps

Por uma questão de otimização, Jogos 2D antigos, organizavam o mapa em uma estrutura de grade com células de mesmo tamanho.

Essas células podem ter formatos diferentes:

- Quadrados
- **Hexagonais**
- Isométricos

Desenhando Mapas

Até agora, vimos como desenhar os objetos dinâmicos por meio de animações via spritesheets, mas e o mapa do jogo, ou seja, os objetos estáticos que definem o ambiente do jogo?

Tilemaps

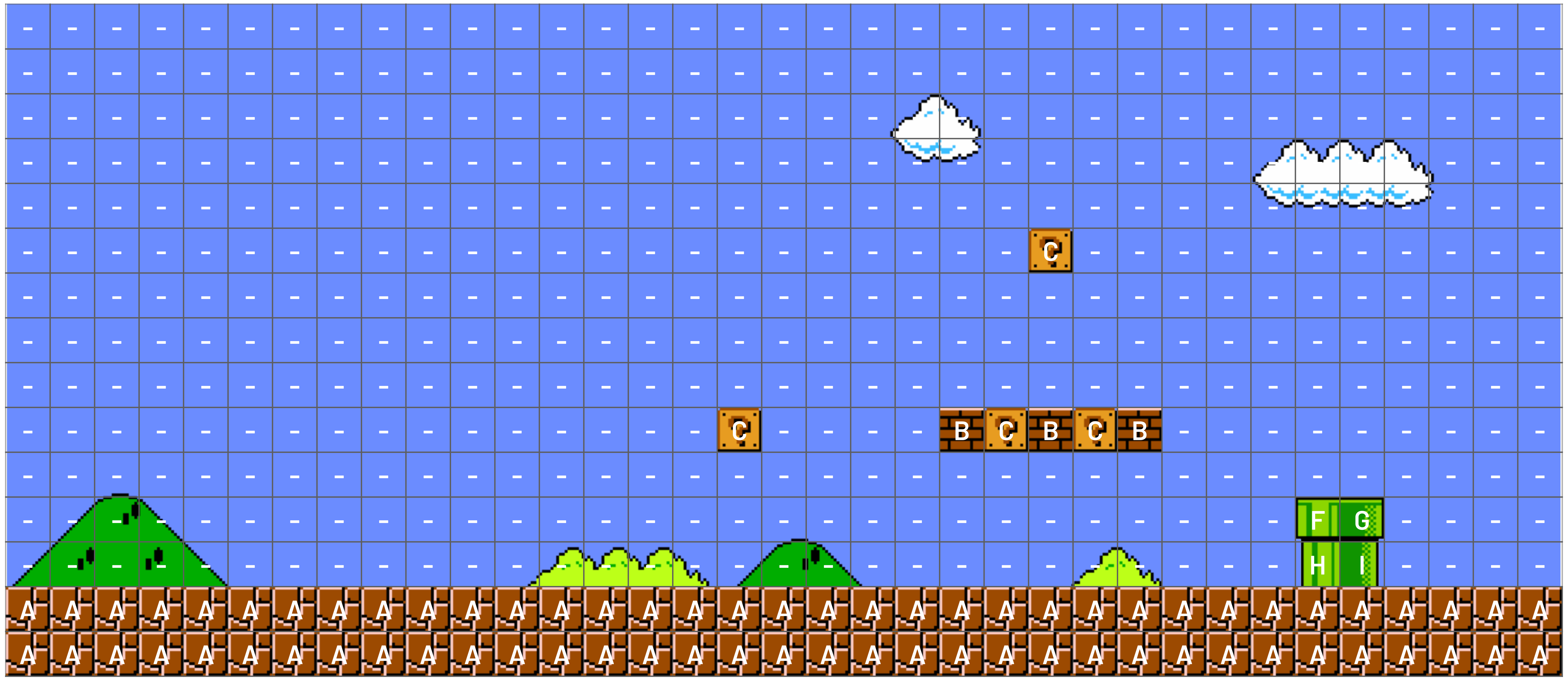
Por uma questão de otimização, Jogos 2D antigos, organizavam o mapa em uma estrutura de grade com células de mesmo tamanho.

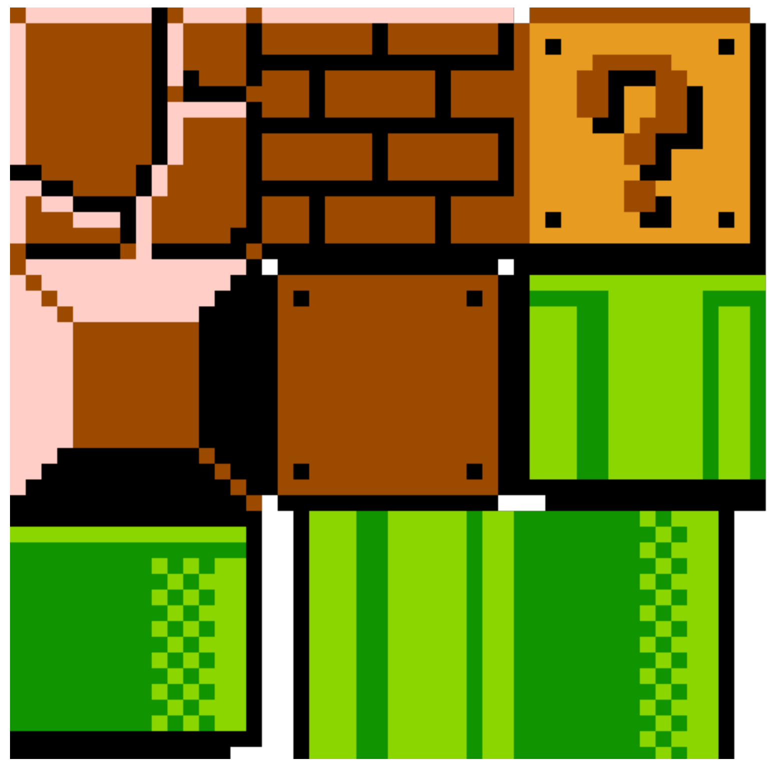
Essas células podem ter formatos diferentes:

- Quadrados
- Hexagonais
- **Isométricos**

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.



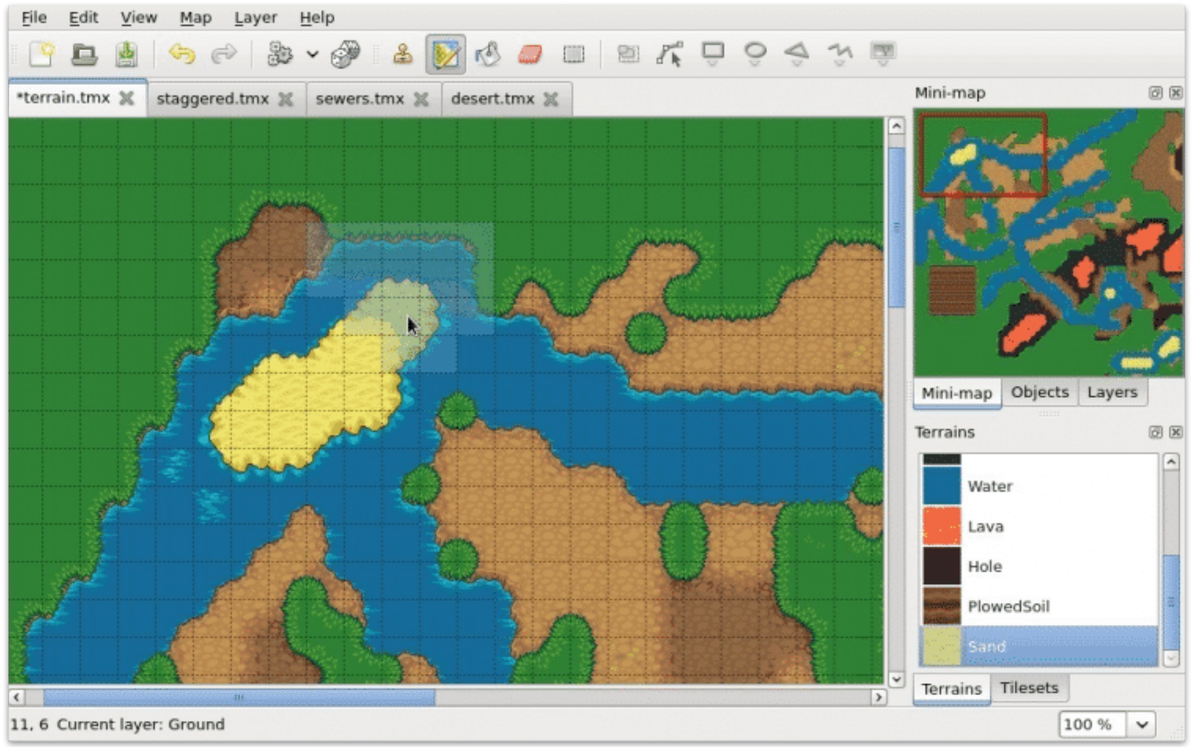


A estrutura de grade dos tilemaps facilita a editação de níveis, pois a posição dos tiles é limitado a coordenadas discretas.

Criando Tilemaps

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

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



Desenhando Mapas

Os jogos 2D modernos não têm mais a mesma limitação de recursos computacionais que os jogos antigos, por isso eles podem desenhar cenas mais detalhadas:

Composição Manual

Na composição manual, cada objeto do mapa é posicionado manualmente, sem restrições de posição.

- As camadas do mapa, backgrounds, midgrounds, e foregrounds, são compostas por grandes imagens.
- Plataformas e paredes são dispostas como blocos desenhados à mão, não como células de uma grade.
- Isso dá ao jogo uma estética mais fluida e orgânico, em vez de repetições em blocos.

Próxima aula

A12: Câmeras 2D

- Representação de Câmeras 2D
- Movimentação de Câmeras 2D
  - Suavização de Movimento
- Efeitos de câmeras 2D
  - Paralaxe
  - Chacoalhão (*Shake*)