
DOOM
How I ported DOOM to the browser with .Net's Blazor WASM
.NET

Speaker

DevRel @ Worldline

Teacher

Yassine Benabbas

yostane

YCoding

Game porting

- Make a game run in platforms other than its original ones
- Achieved by adapting the code for the new platform
- Not porting: virtual machine or emulator

Game porting

- Make a game run in platforms other than its original ones
- Achieved by adapting the code for the new platform
- Not porting: virtual machine or emulator

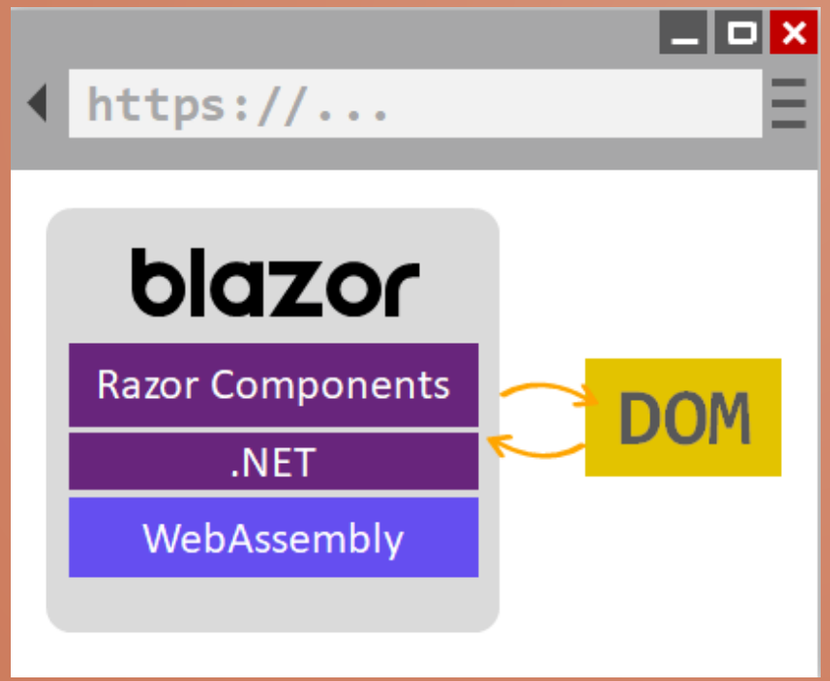
MVG's video is a great source of inspiration

.net and the browser

- .net: OSS cross-platform framework
- C# language

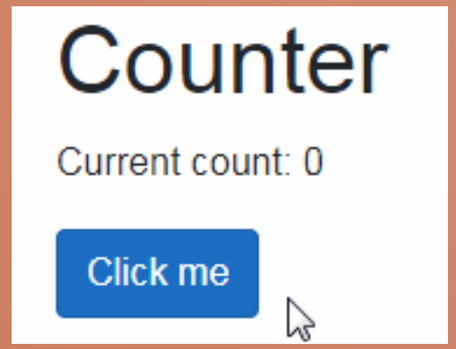
.net and the browser

- .net: OSS cross-platform framework
- C# language
- In 2020 .Net 5 introduced Blazor WASM
  - Component based
  - We can run .net locally on the browser



A Razor component

```
 1 @page "/counter"
 2
 3 <h1>Counter</h1>
 4 <p>Current count: @currentCount</p>
 5 <button @onclick="IncrementCount">Click me</button>
 6
 7 @code {
 8     private int currentCount = 0;
 9     private void IncrementCount() { currentCount++; }
10 }
```



Let's port a .net game !

- Released in 1993 for DOS
- One the most successful First Person Shooters
- Has two parts:
  - **engine: Game logic**
  - WAD file: all assets and maps

Doom is portable by design

Ported to a LOT of platforms

http://mrglitchsreviews.blogspot.com/2012/09/doom-console-ports.html

https://www.link-cable.com/top-10-weird-doom-ports/

ManagedDoom

- **.Net** Port of LinuxDoom
- ManagedDoom V1 Uses SFML (graphics + audio + input)
  - V2 (in beta as of Jan 3) uses silk.net
- My work is a based on **ManagedDoom V1**

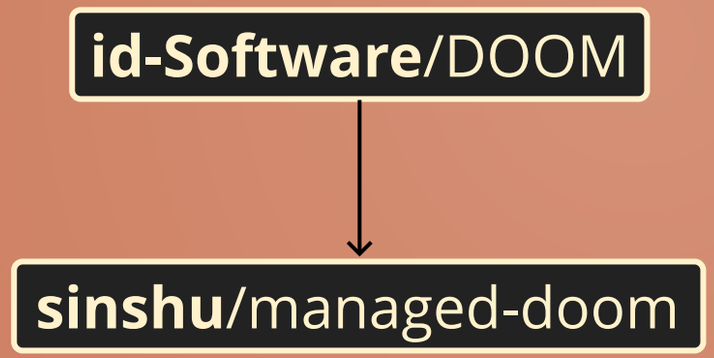
ManagedDoom

- **.Net** Port of LinuxDoom
- ManagedDoom V1 Uses SFML (graphics + audio + input)
  - V2 (in beta as of Jan 3) uses silk.net
- My work is a based on **ManagedDoom V1**

id-Software/DOOM

ManagedDoom

- **.Net** Port of LinuxDoom
- ManagedDoom V1 Uses SFML (graphics + audio + input)
  - V2 (in beta as of Jan 3) uses silk.net
- My work is a based on **ManagedDoom V1**



ManagedDoom

- **.Net** Port of LinuxDoom
- ManagedDoom V1 Uses SFML (graphics + audio + input)
  - V2 (in beta as of Jan 3) uses silk.net
- My work is a based on **ManagedDoom V1**

id-Software/DOOM

↓

sinshu/managed-doom

↓

yostane/MangedDoom-Blazor

Porting strategy

Porting strategy

- Compile the app: replace SFML code with "*TODO: implement*"

Porting strategy

- Compile the app: replace SFML code with "*TODO: implement*"
- Implement TODOs little by little, priority to frame rendering

Porting strategy

- Compile the app: replace SFML code with "*TODO: implement*"
- Implement TODOs little by little, priority to frame rendering
- Optimize and clean later strategy

Porting strategy

- Compile the app: replace SFML code with "*TODO: implement*"
- Implement TODOs little by little, priority to frame rendering
- Optimize and clean later strategy
- Read Doom-Wiki and SFML documentation *only when necessary*
  - Used to understand frame format

Porting strategy

- Compile the app: replace SFML code with "*TODO: implement*"
- Implement TODOs little by little, priority to frame rendering
- Optimize and clean later strategy
- Read Doom-Wiki and SFML documentation *only when necessary*
  - Used to understand frame format

2 weeks

as a side-project

Game loop pseudo-code

```
1 while (waitForNextFrame()){
2   const input = getPlayerInput();
3   const { frame, audio }
4               = UpdateGameState(input, WAD);
5   render(frame);
6   play(audio);
7 }
```

Game loop pseudo-code

```
1 while (waitForNextFrame()){
2    const input = getPlayerInput();
3    const { frame, audio }
4                = UpdateGameState(input, WAD);
5    render(frame);
6    play(audio);
7 }
```

Game loop pseudo-code

```
1 while (waitForNextFrame()){
2    const input = getPlayerInput();
3    const { frame, audio }
4                = UpdateGameState(input, WAD);
5    render(frame);
6    play(audio);
7 }
```

Game loop pseudo-code

```
1 while (waitForNextFrame()){
2   const input = getPlayerInput();
3   const { frame, audio }
4                 = UpdateGameState(input, WAD);
5   render(frame);
6   play(audio);
7 }
```

Game loop pseudo-code

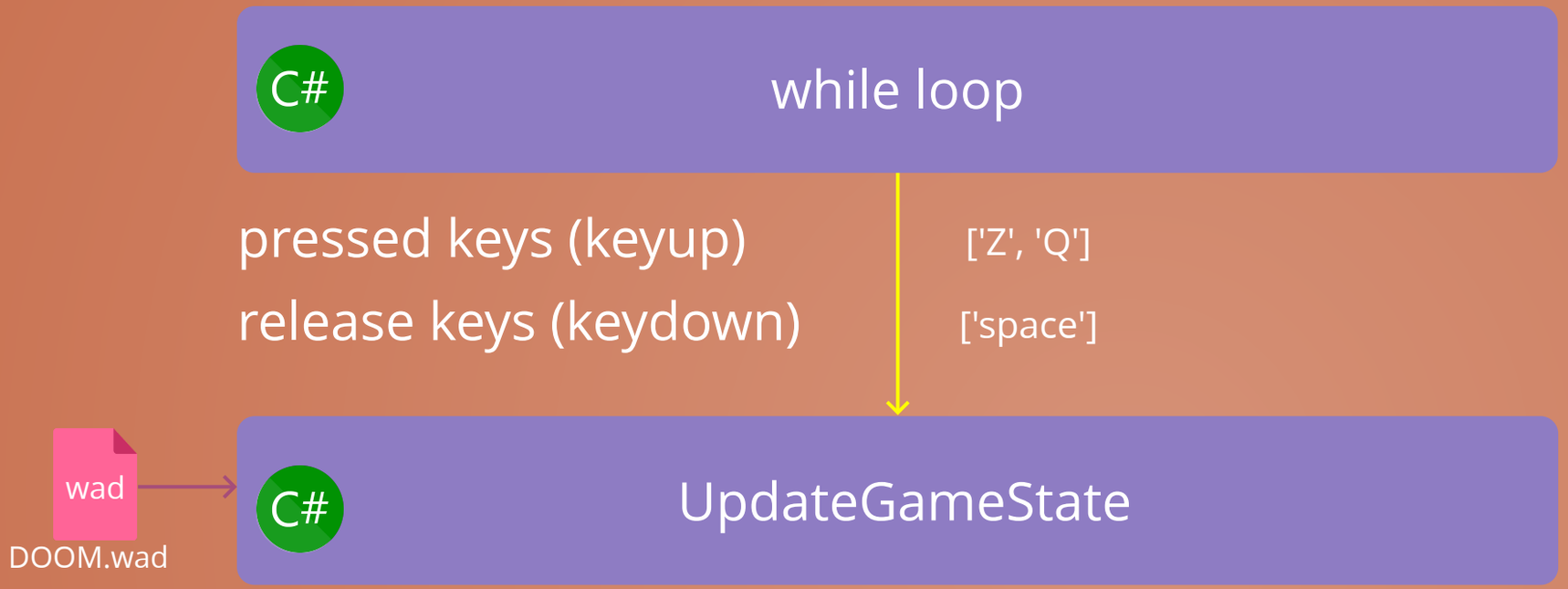
```
1 while (waitForNextFrame()){
2    const input = getPlayerInput();
3    const { frame, audio }
4                = UpdateGameState(input, WAD);
5    render(frame);
6    play(audio);
7 }
```

Game loop pseudo-code

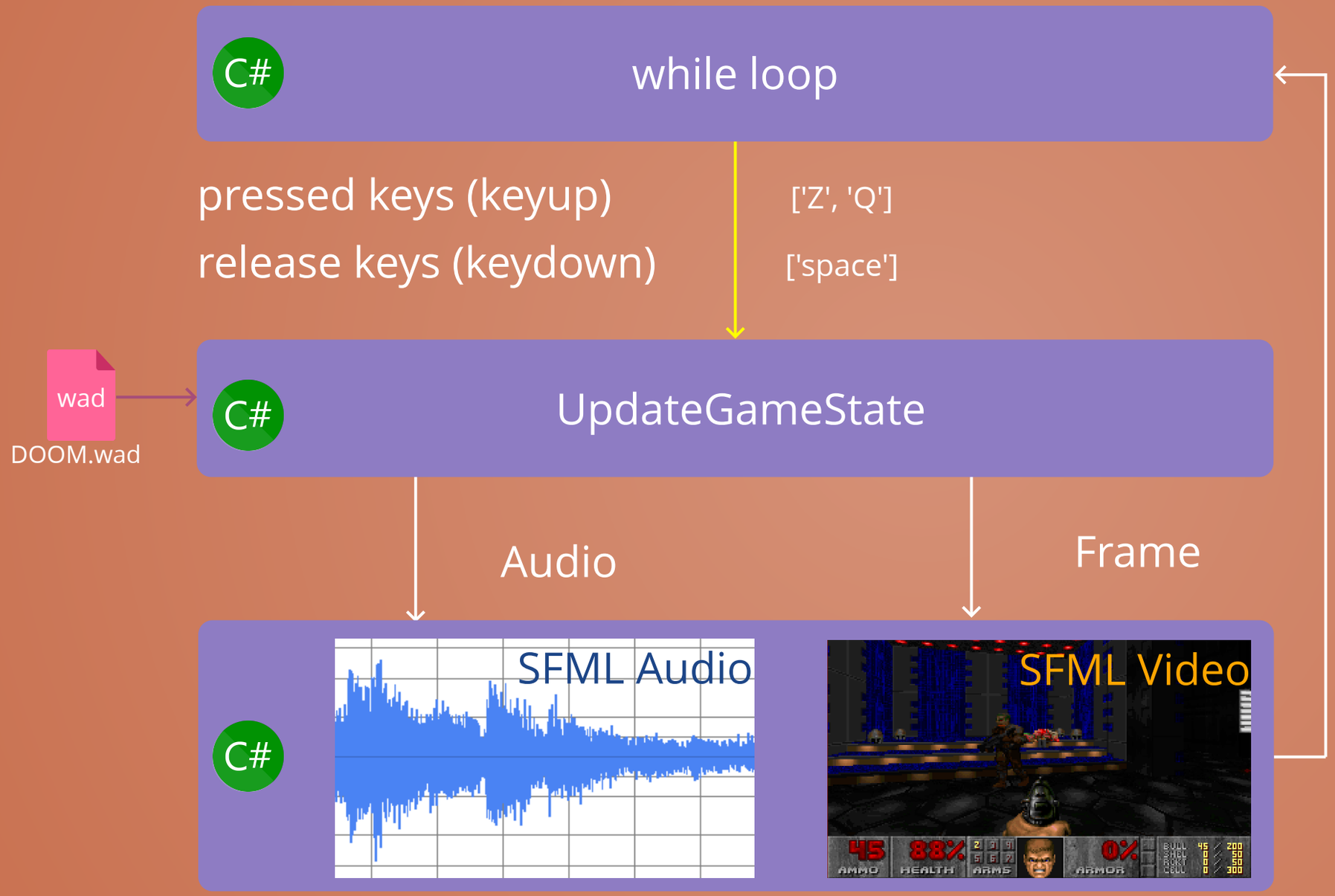
```
1 while (waitForNextFrame()){
2   const input = getPlayerInput();
3   const { frame, audio }
4               = UpdateGameState(input, WAD);
5   render(frame);
6   play(audio);
7 }
```

ManagedDoom V1 architecture

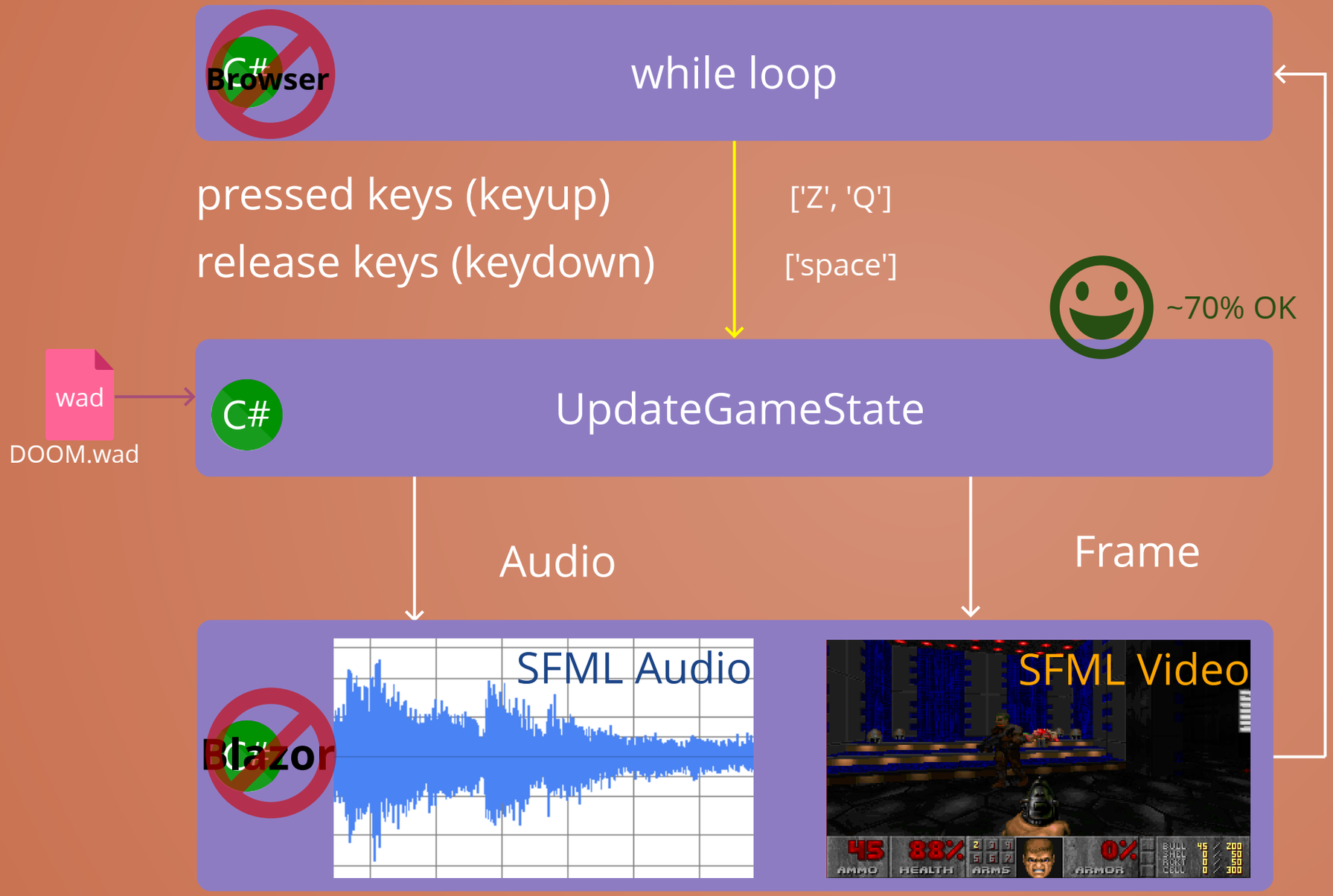
ManagedDoom V1 architecture



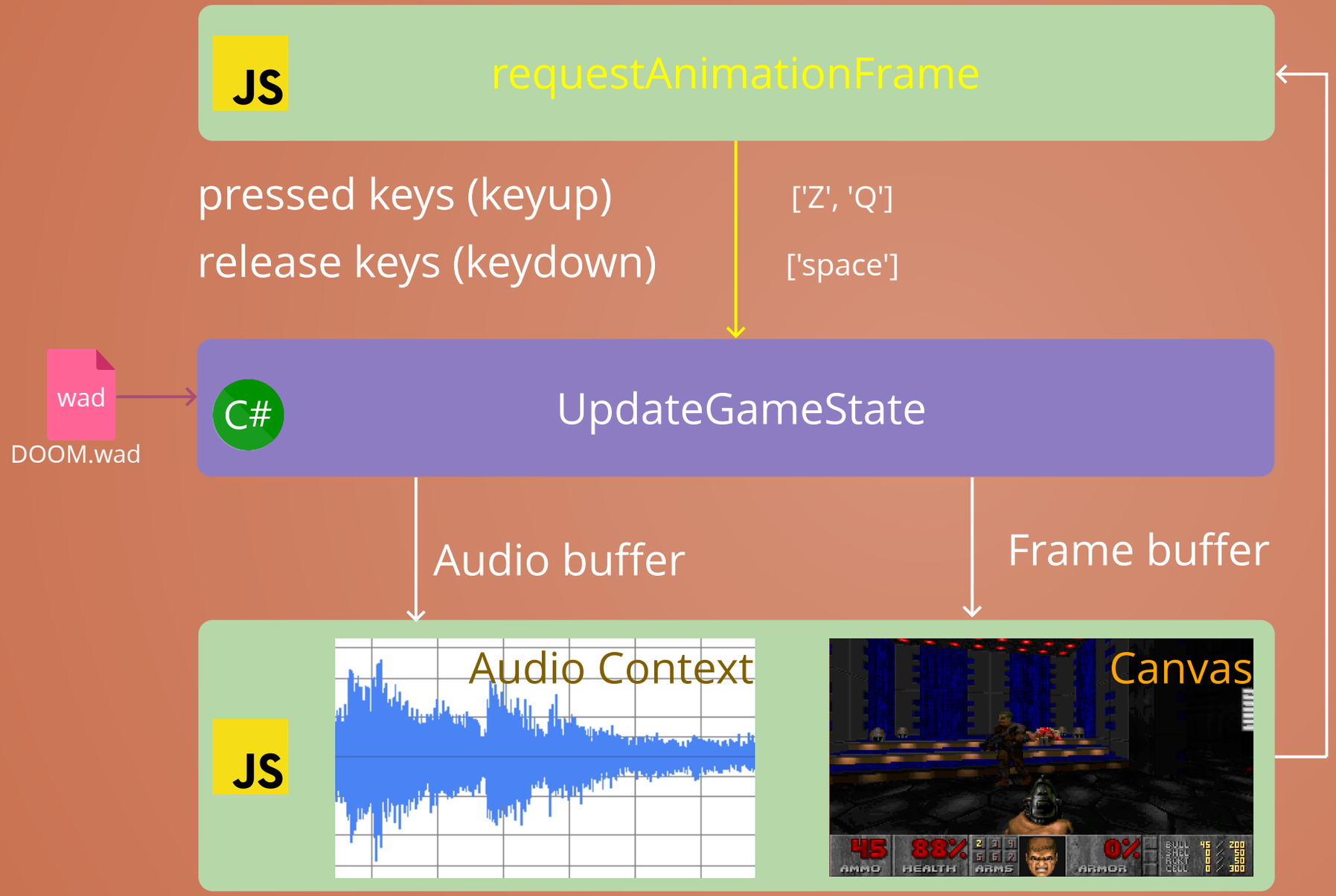
ManagedDoom V1 architecture



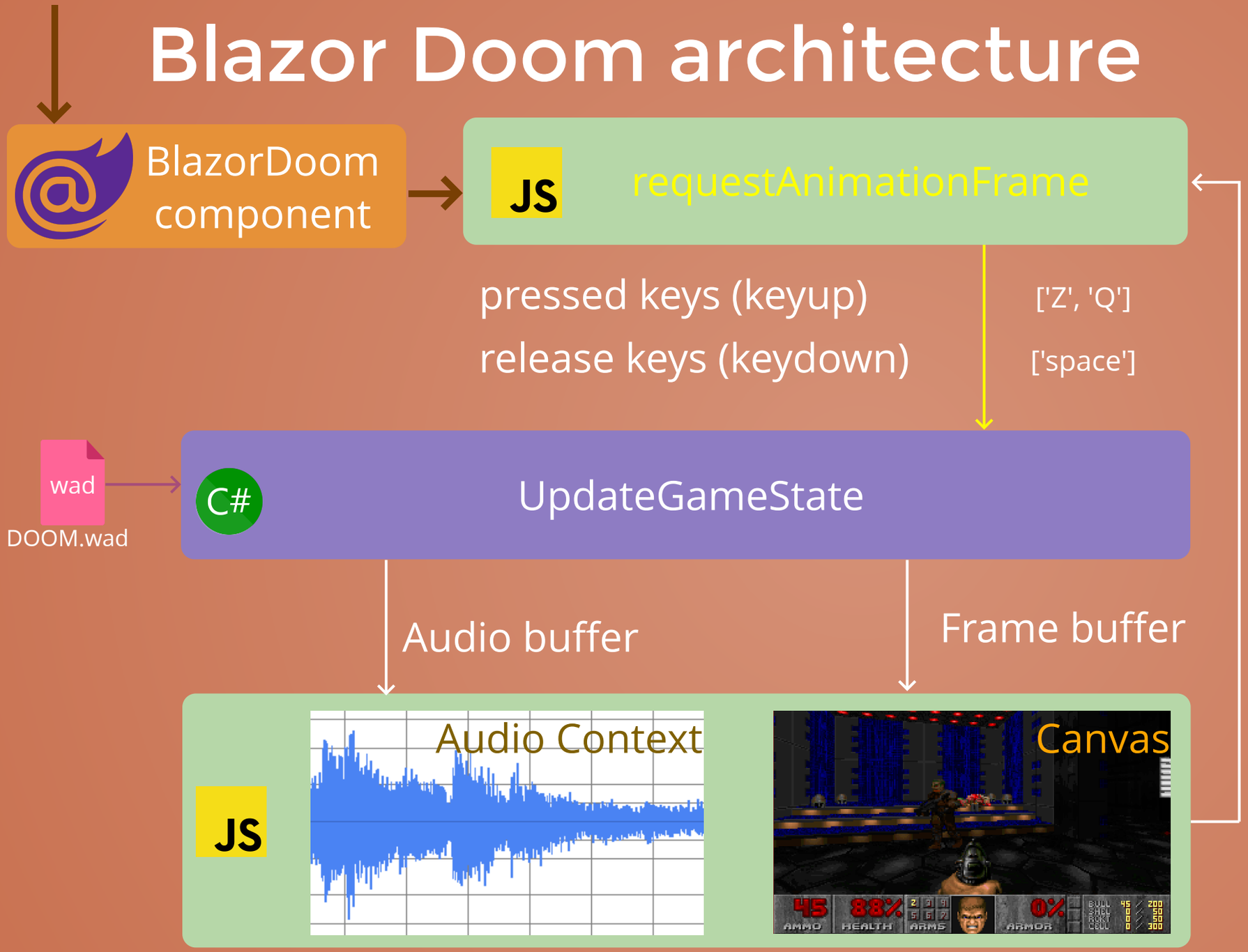
ManagedDoom V1 architecture



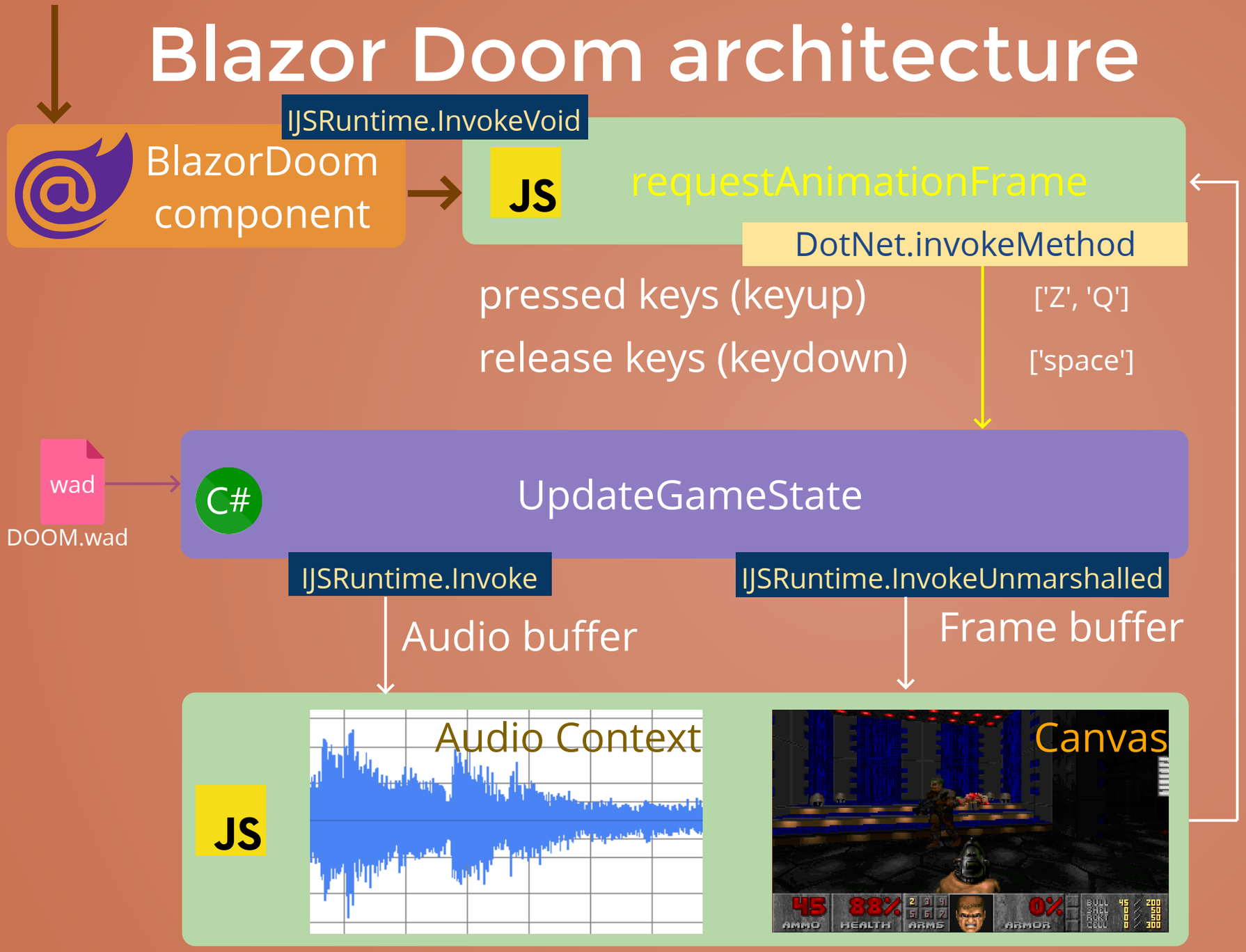
Blazor Doom architecture



Blazor Doom architecture



Blazor Doom architecture



Technical details

With some code !

https://doom.fandom.com/wiki/Cacodemon/Doom

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```razor
<canvas id="canvas" image-rendering: pixelated;" ...>
</canvas>
@code {
  // Entry point of the game
  private async Task StartGame()
  {
    // steup the game object
    app = new ManagedDoom.DoomApplication(...);
    // JS method that calls "requestAnimationFrame"
    jsProcessRuntime.InvokeVoid("gameLoop");
  }
}
```

C#

```javascript
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

JS

Entry point and frame pacing

```js
window.gameLoop = function (timestamp) {
  // Check the pacing
  if (timestamp - lastFrameTimestamp >= frameTime) {
    lastFrameTimestamp = timestamp;
    // Updates the game state (advances a frame)
    DotNet.invokeMethod('BlazorDoom', 'UpdateGame', downKeys, upKeys)
  }
  // This replaces the for loop in a traditional game
  // Request the browser to notify us when we do the next iteration
  window.requestAnimationFrame(window.gameLoop);
}
```

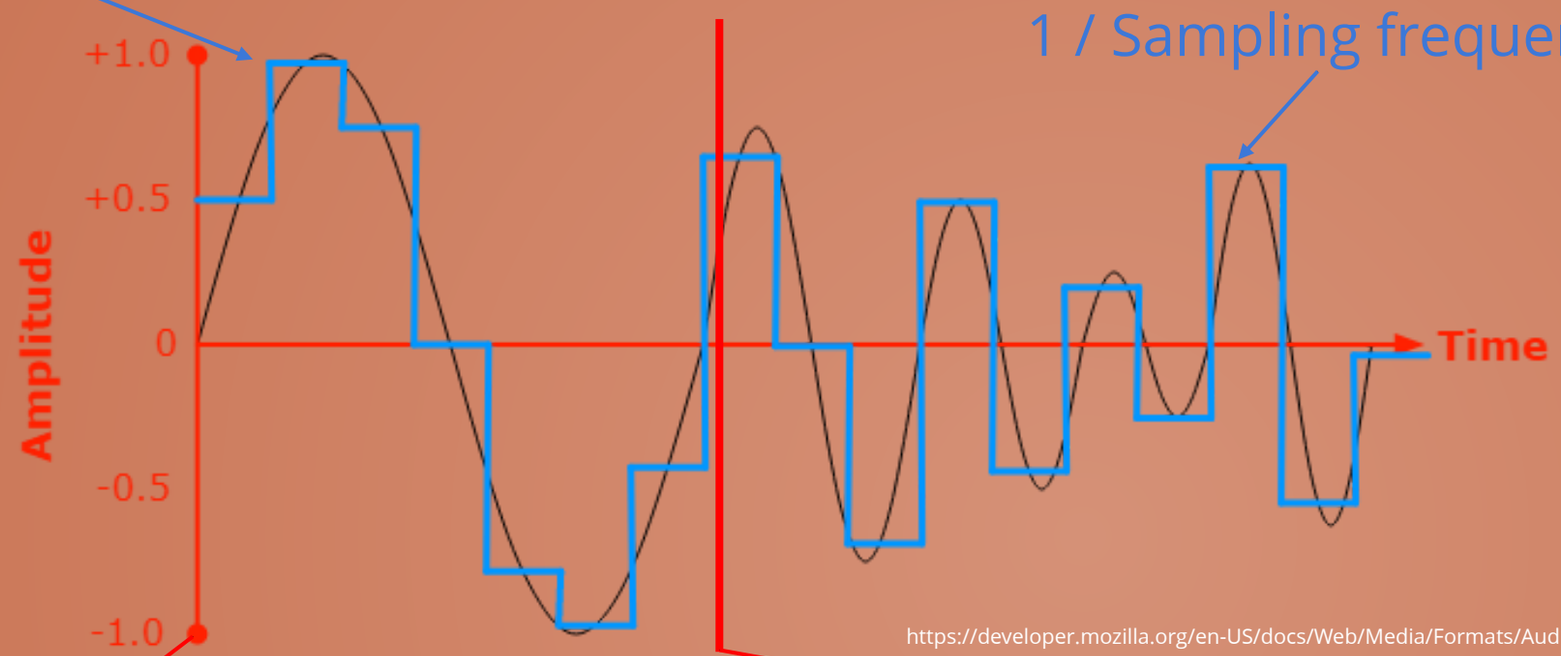

```cs
// The code that I showed earlier
[JSInvokable("GameLoop")]
public static void UpdateGame(uint[] downKeys, uint[] upKeys)
{
  app.Run(downKeys, upKeys);
}
```

Audio and video rendering

Audio playback

https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

Audio playback



https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

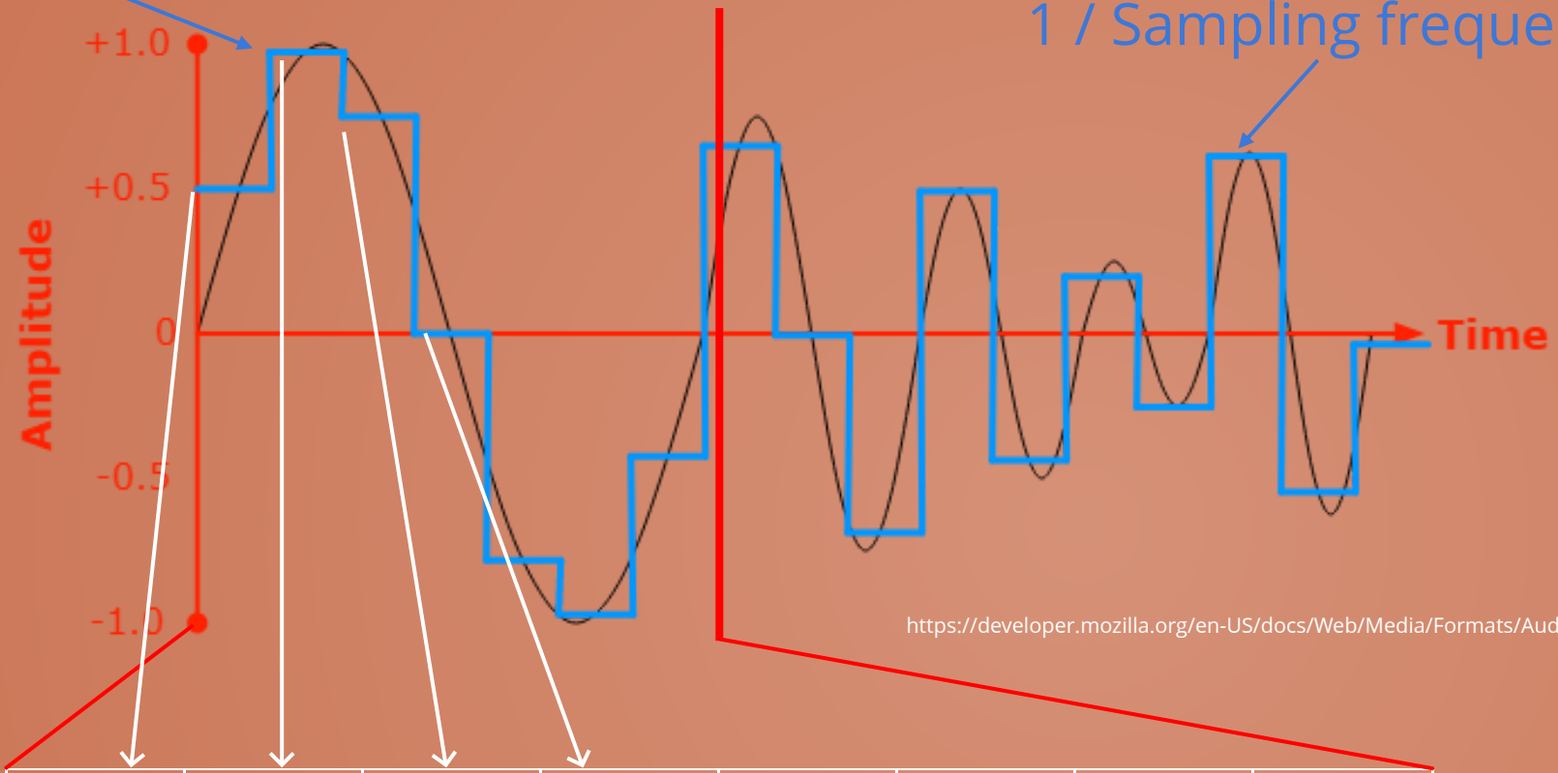
C#

0.5	1	0.75	0	-0.75	-1	-0.5	0

Audio playback

Sample

1 / Sampling frequency



https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

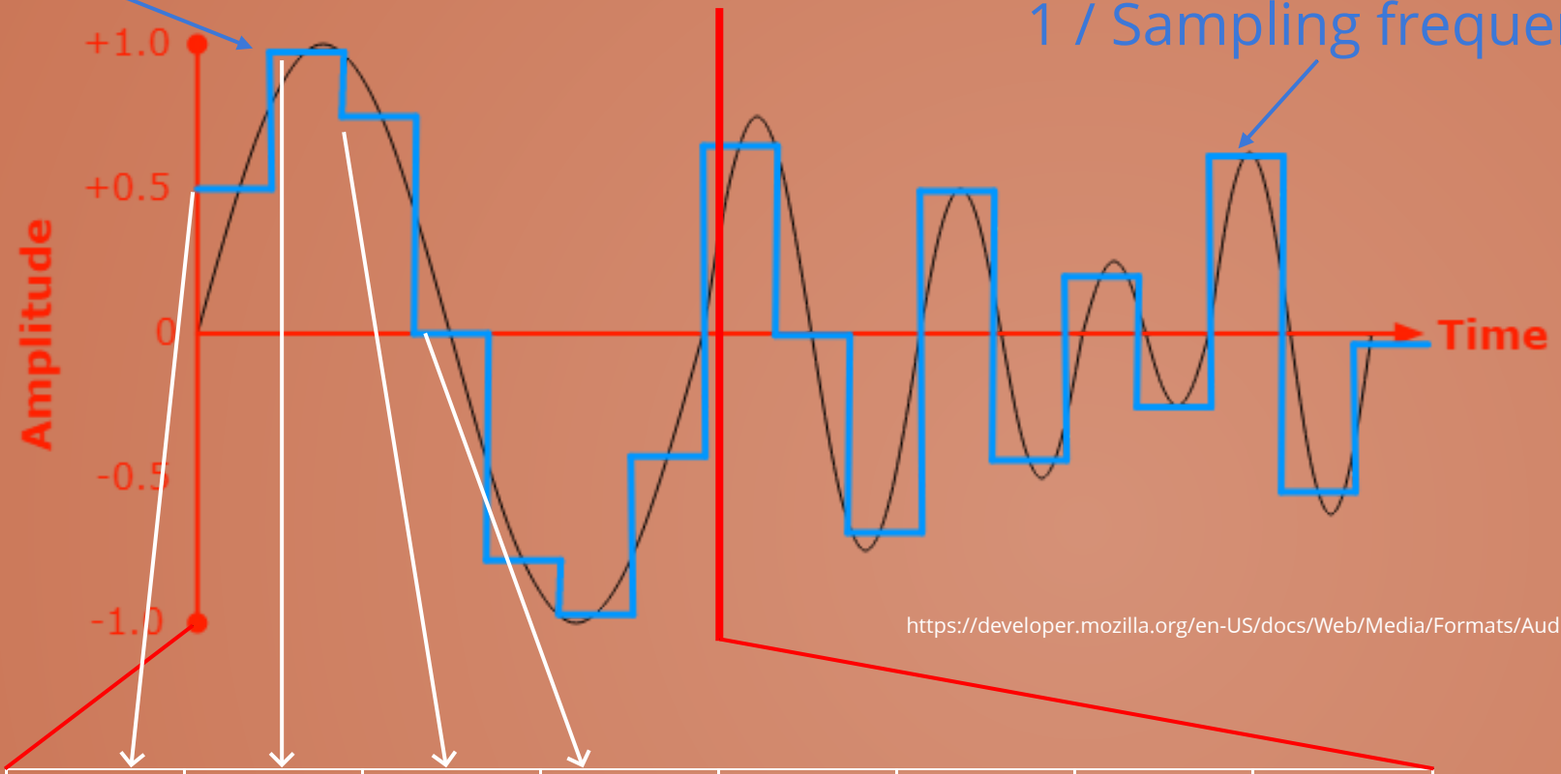
C#

0.5	1	0.75	0	-0.75	-1	-0.5	0

Audio playback

Sample

1 / Sampling frequency



https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

C#

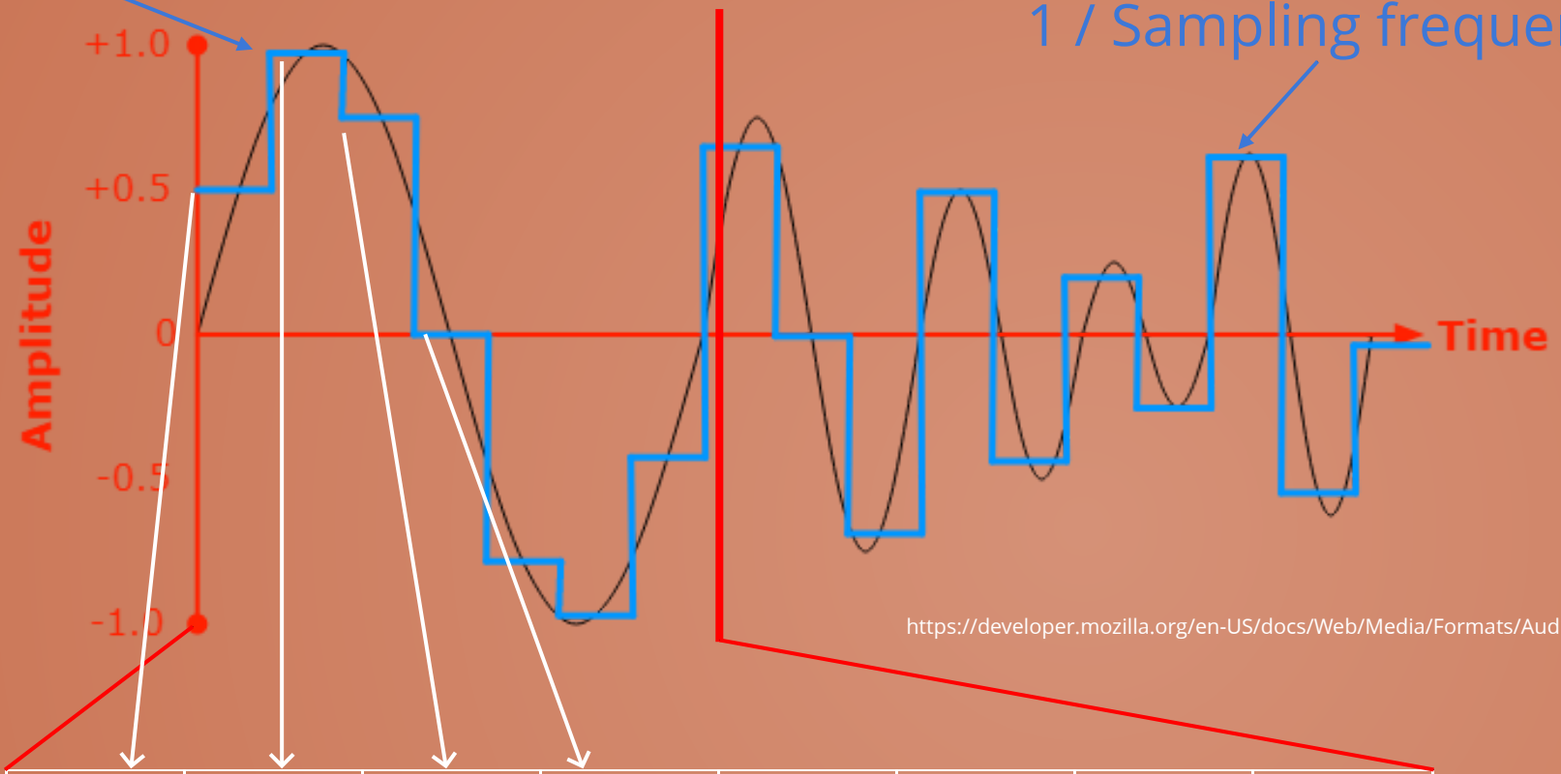
0.5	1	0.75	0	-0.75	-1	-0.5	0

+ Sampling frequency

Audio playback

Sample

1 / Sampling frequency



https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

C#

0.5	1	0.75	0	-0.75	-1	-0.5	0

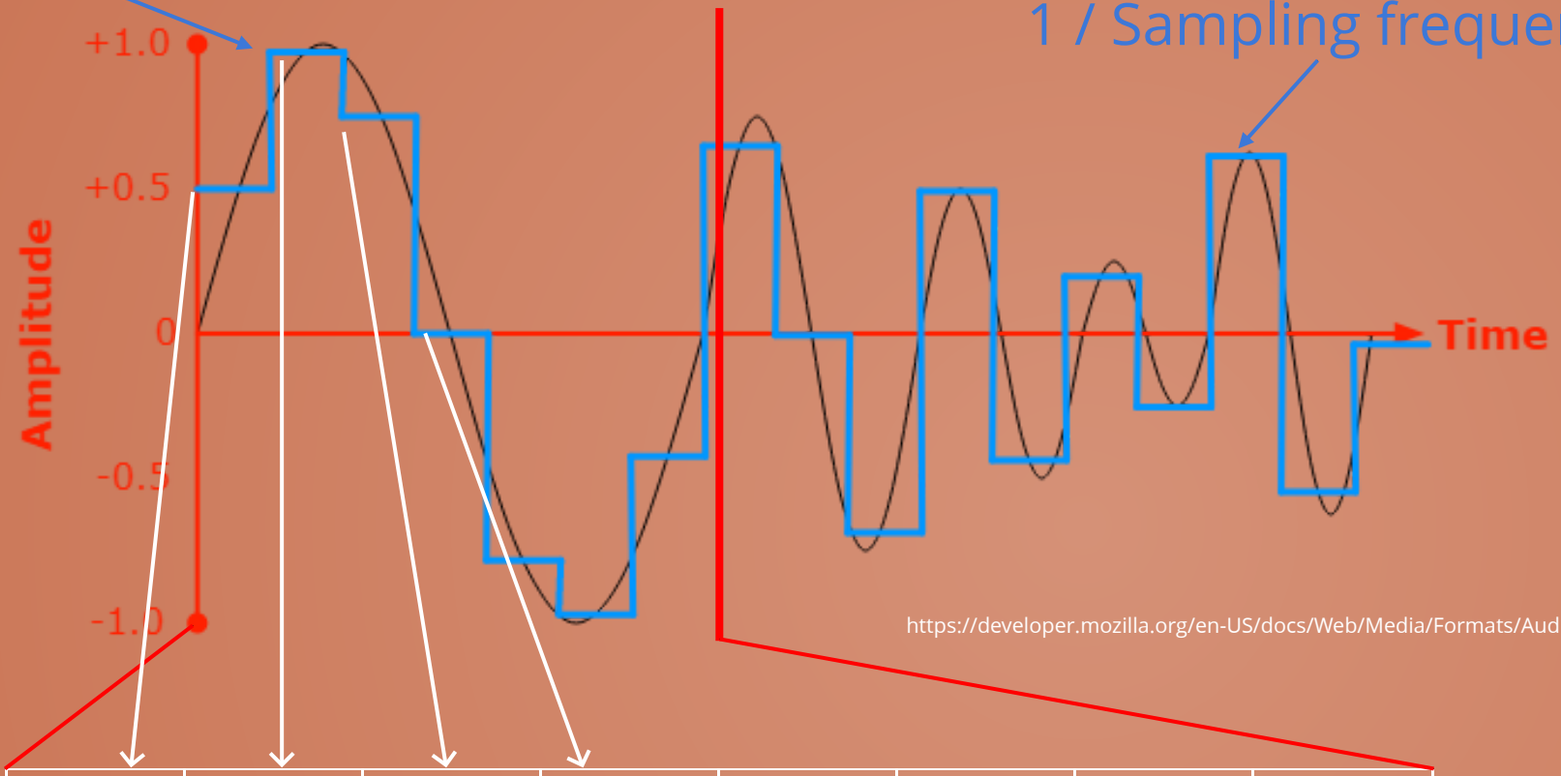
+ Sampling frequency

JS AudioContext

Audio playback

Sample

1 / Sampling frequency



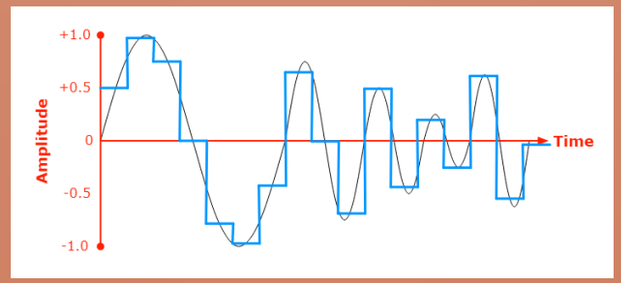
https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts

C#

0.5	1	0.75	0	-0.75	-1	-0.5	0

+ Sampling frequency

JS AudioContext



Audio playback

```
// Somewhere in the Doom Engine's audio module
DoomApplication.WebAssemblyJSRuntime.Invoke<object>(
        "playSound", new object[] { samples, sampleRate, 0, Position }
);
```


```
playByteArray(samples, sampleRate) {
  const audioBuffer = this.context.createBuffer(
    1, length, this.context.sampleRate
  );
  var channelData = audioBuffer.getChannelData(0);
  // JS receives a weird "samples" array
  for (let i = 0; i < length; i += 2) {
    // Scale the value to between -1 and 1
    channelData[i] = samples[i] / 0xffff;
  }
  // Play the audio
  var source = this.context.createBufferSource();
  source.buffer = audioBuffer;
  source.connect(this.context.destination);
  source.start();
}
```

Audio playback

```csharp
// Somewhere in the Doom Engine's audio module
DoomApplication.WebAssemblyJSRuntime.Invoke<object>(
        "playSound", new object[] { samples, sampleRate, 0, Position }
);
```


```javascript
playByteArray(samples, sampleRate) {
  const audioBuffer = this.context.createBuffer(
    1, length, this.context.sampleRate
  );
  var channelData = audioBuffer.getChannelData(0);
  // JS receives a weird "samples" array
  for (let i = 0; i < length; i += 2) {
    // Scale the value to between -1 and 1
    channelData[i] = samples[i] / 0xffff;
  }
  // Play the audio
  var source = this.context.createBufferSource();
  source.buffer = audioBuffer;
  source.connect(this.context.destination);
  source.start();
}
```

Audio playback

```
// Somewhere in the Doom Engine's audio module
DoomApplication.WebAssemblyJSRuntime.Invoke<object>(
        "playSound", new object[] { samples, sampleRate, 0, Position }
);
```

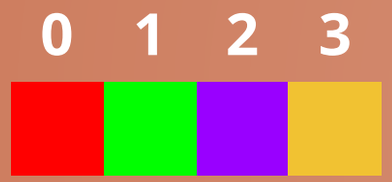

```
playByteArray(samples, sampleRate) {
  const audioBuffer = this.context.createBuffer(
    1, length, this.context.sampleRate
  );
  var channelData = audioBuffer.getChannelData(0);
  // JS receives a weird "samples" array
  for (let i = 0; i < length; i += 2) {
    // Scale the value to between -1 and 1
    channelData[i] = samples[i] / 0xffff;
  }
  // Play the audio
  var source = this.context.createBufferSource();
  source.buffer = audioBuffer;
  source.connect(this.context.destination);
  source.start();
}
```

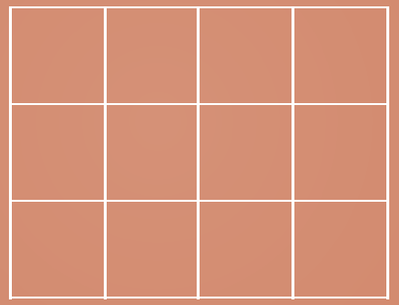
From 1D frame to a 2D frame

Frame data

0	1	2	3	1	1	2	3	2	2	0	1

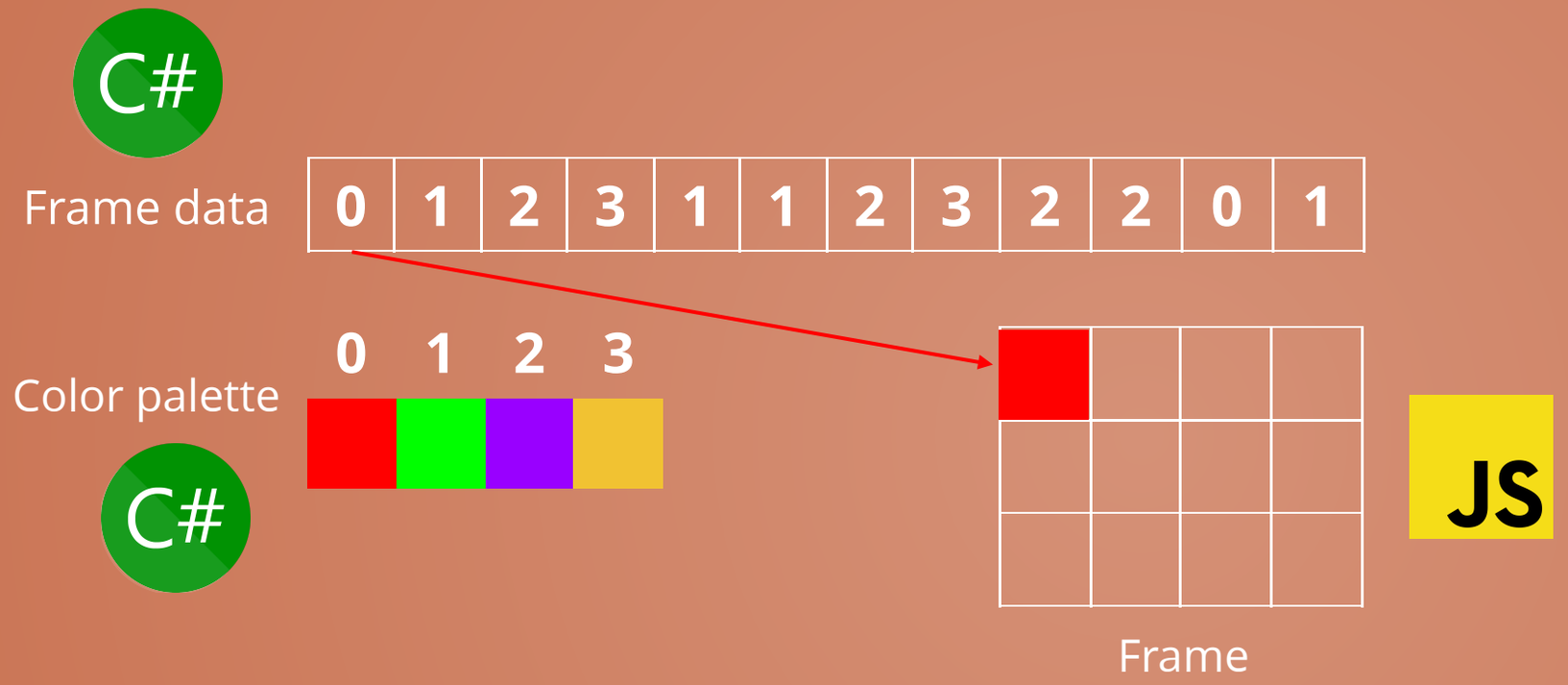
Color palette



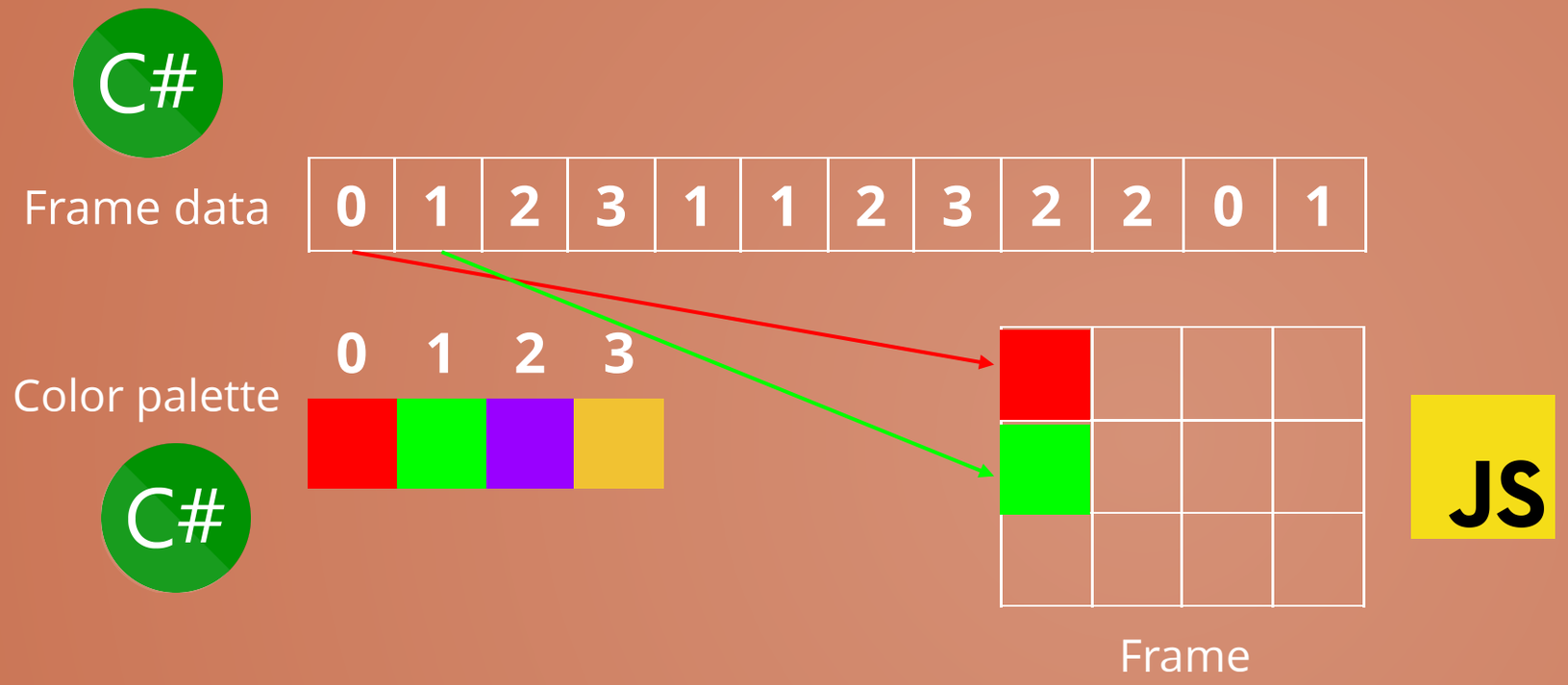


Frame

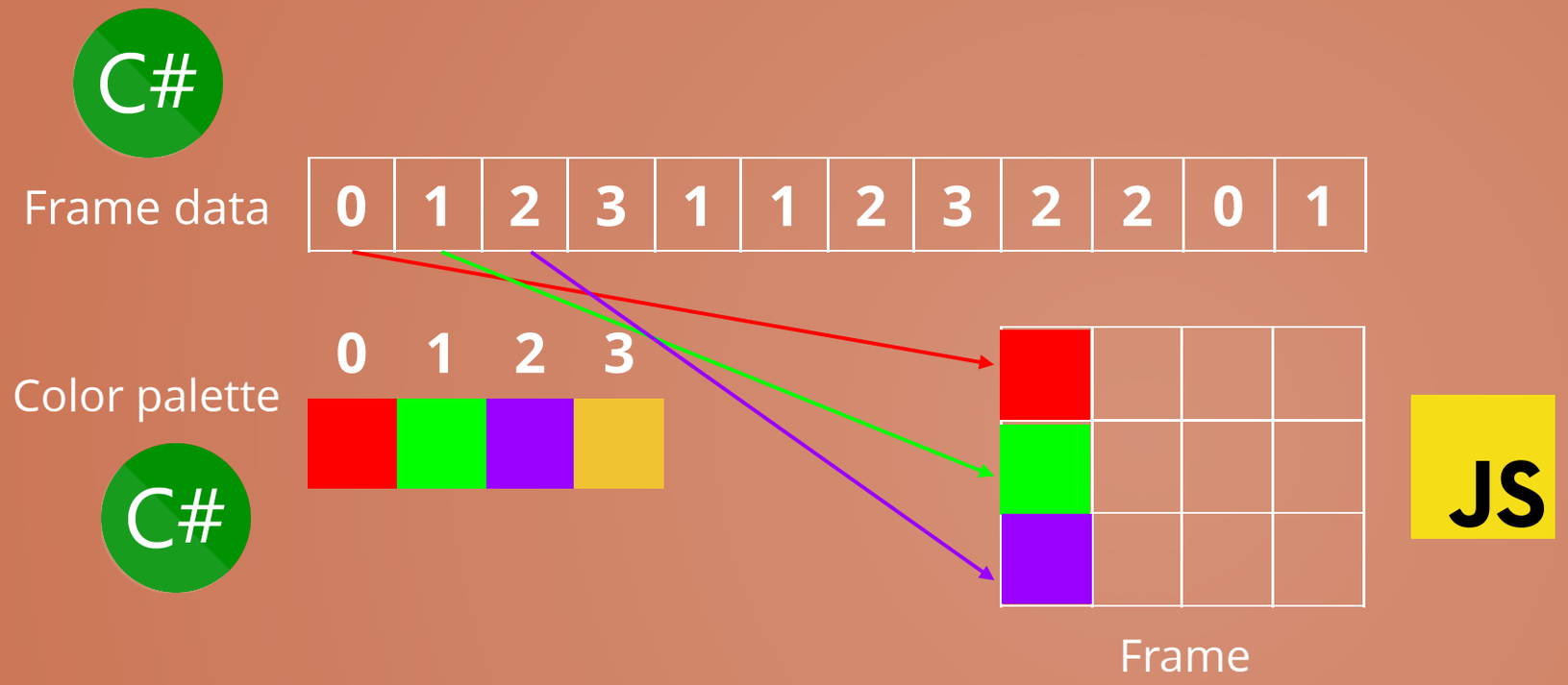
From 1D frame to a 2D frame



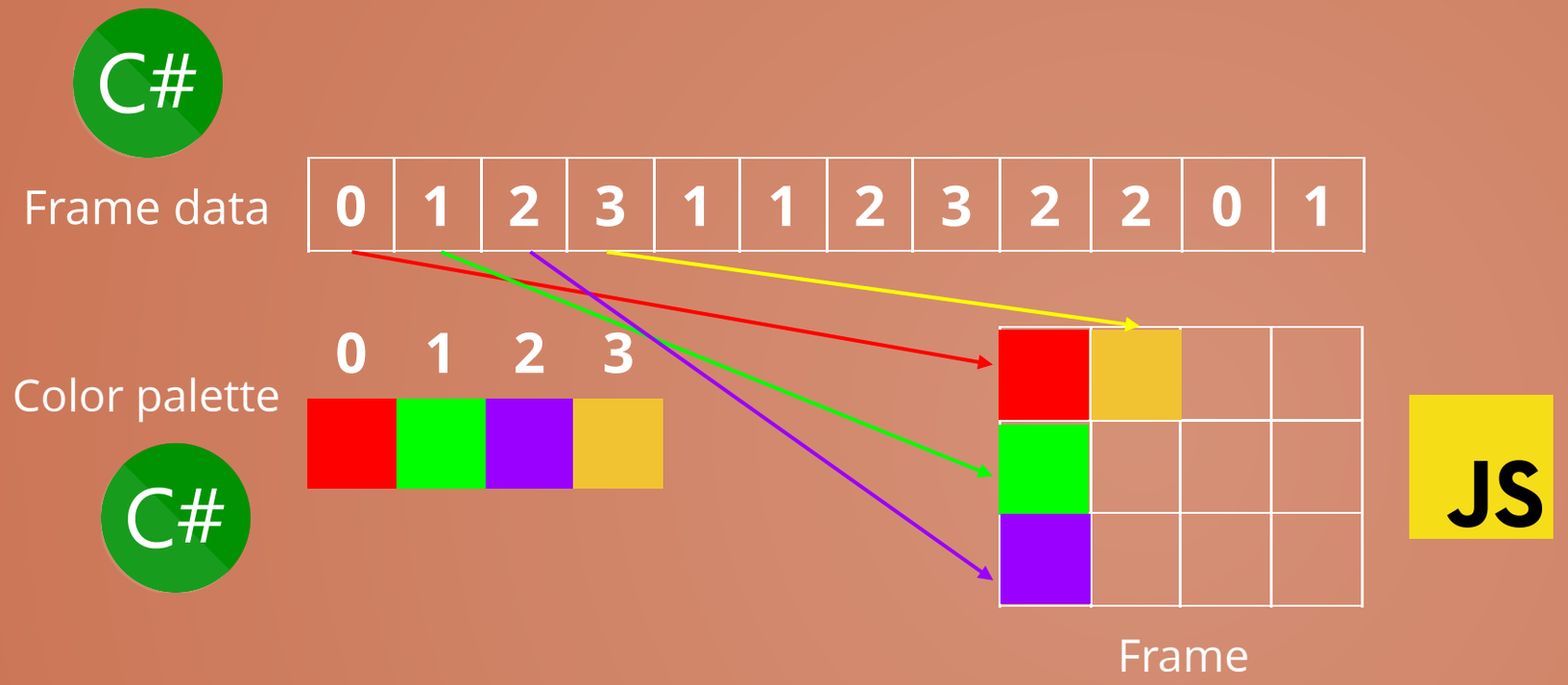
From 1D frame to a 2D frame



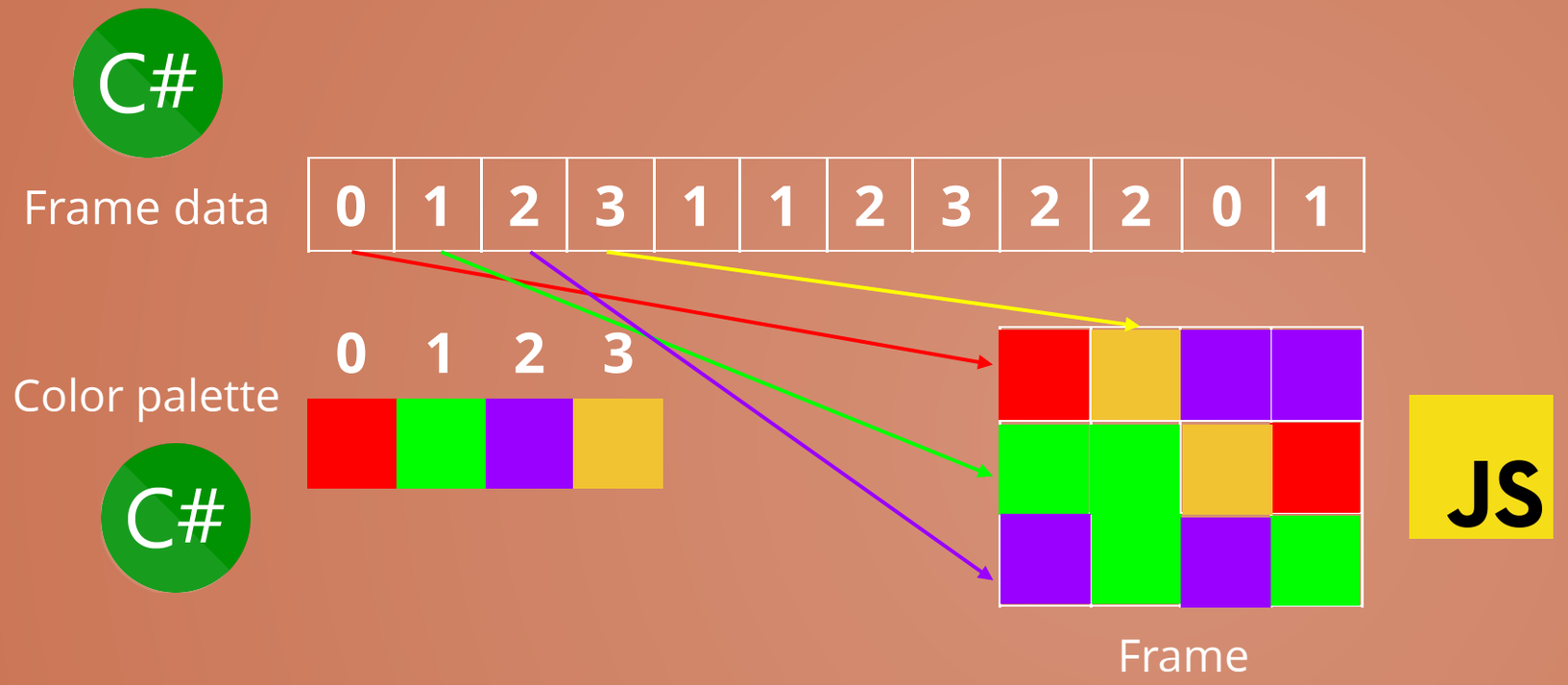
From 1D frame to a 2D frame



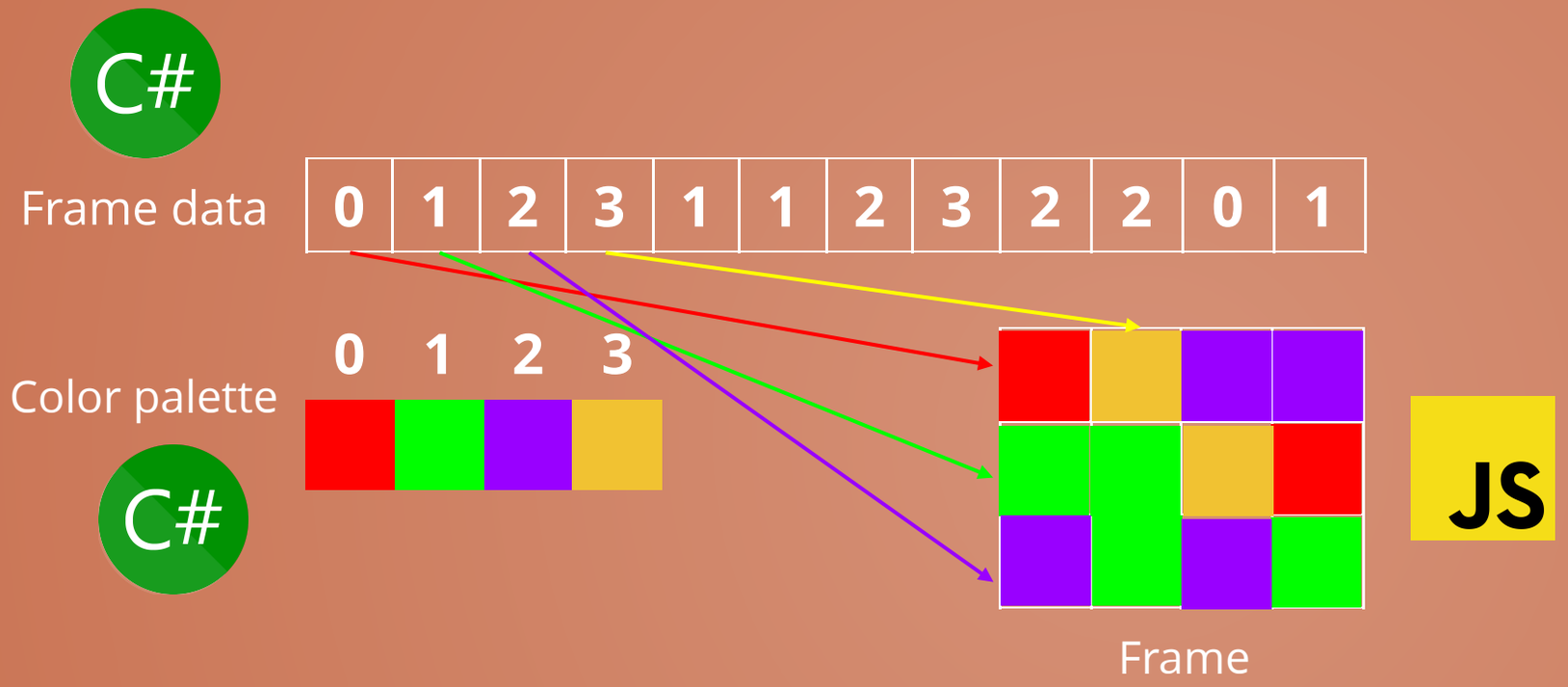
From 1D frame to a 2D frame



From 1D frame to a 2D frame

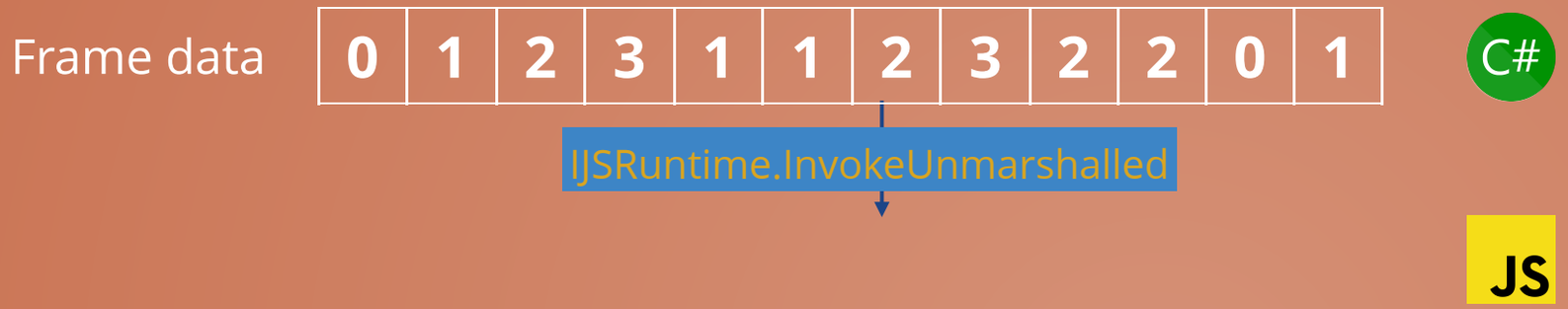


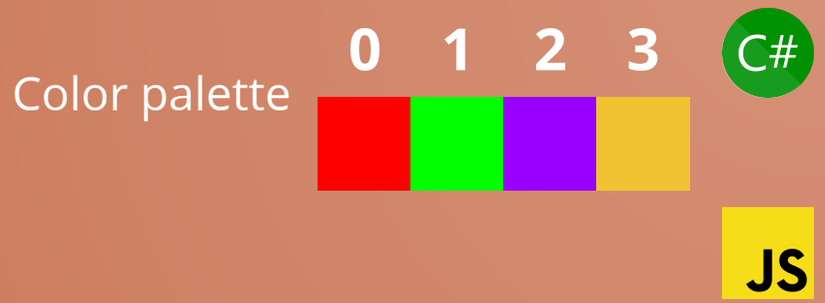
From 1D frame to a 2D frame



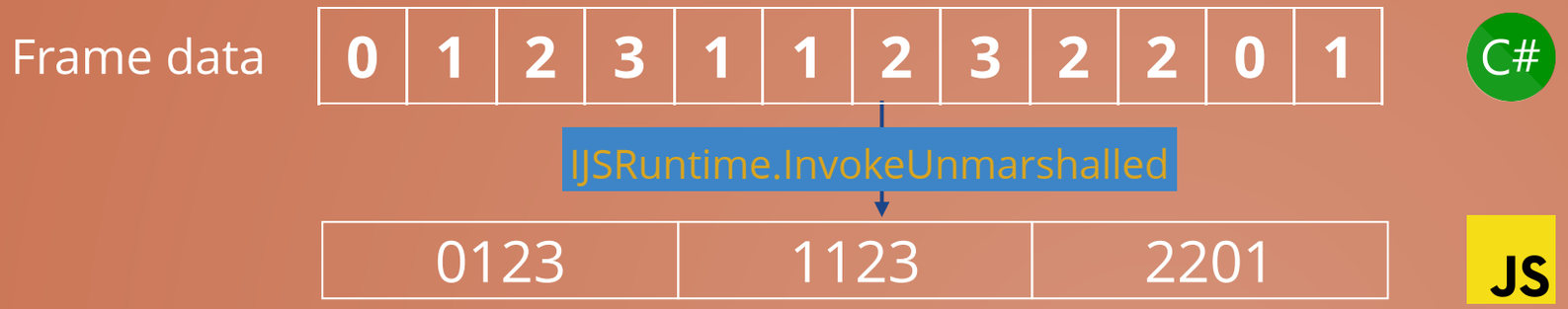
- Image built from top to bottom and from left to right
- Doom uses color indexing

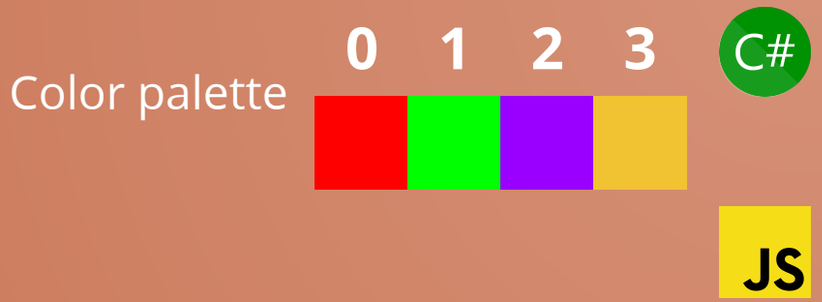
C# Byte Array to JS, considerations



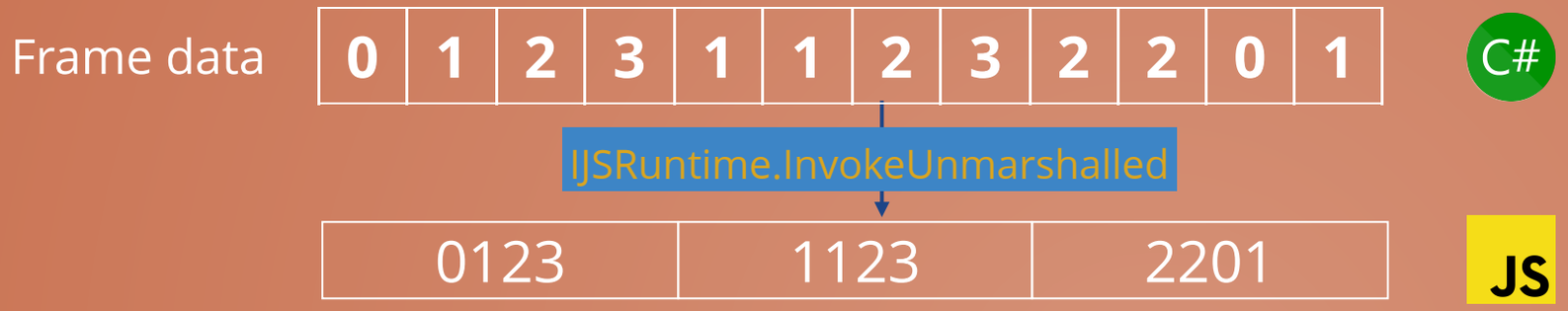


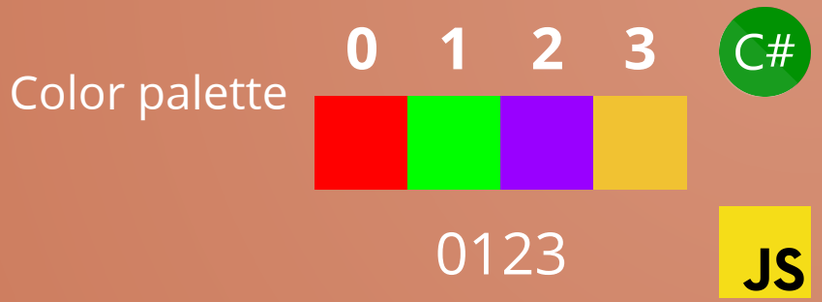
C# Byte Array to JS, considerations



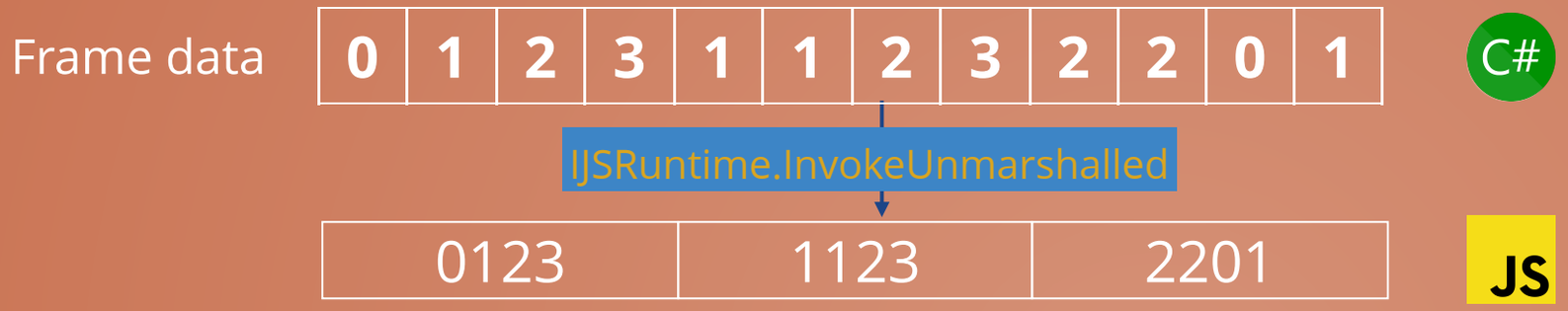


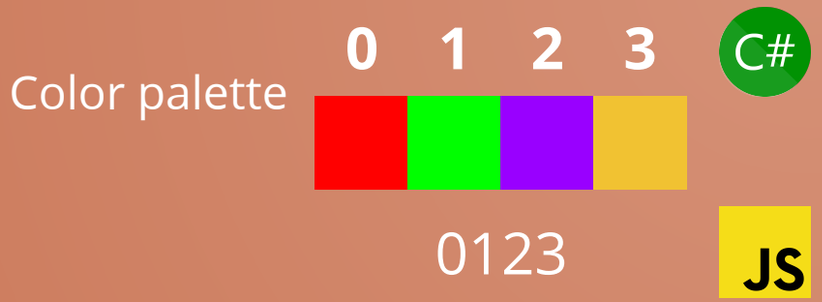
C# Byte Array to JS, considerations





C# Byte Array to JS, considerations





- **4 bytes in C# -> 1 number in JS**
- **n elements in C# -> (n / 4) elements in JS**
- **Bit shifting required in JS !**

Frame rendering

```
1 // Somwhere in the Doom Engine's graphics module
2 var args = new object[] { screen.Data, colors, 320, 200 };
3 // Send the frame buffer to JS
4 DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
5         ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```

Frame rendering

```
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Frame rendering

```csharp
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```javascript
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Frame rendering

```
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Frame rendering

```
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Frame rendering

```
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Frame rendering

```
// Somwhere in the Doom Engine's graphics module
var args = new object[] { screen.Data, colors, 320, 200 };
// Send the frame buffer to JS
DoomApplication.WebAssemblyJSRuntime.InvokeUnmarshalled<byte[], uint[], int>
        ("renderWithColorsAndScreenDataUnmarshalled", screen.Data, colors);
```


```
window.renderWithColorsAndScreenDataUnmarshalled = (screenData, colors) => {
  // JS receives an array with 4 bytes per item
  for (var i = 0; i < (width * height) / 4; i += 1) {
    // Gets the array sent from C#
    const screenDataItem = BINDING.mono_array_get(screenData, i);
    for (var mask = 0; mask <= 24; mask += 8) {
      let dataIndex = y * (width * 4) + x;
      setSinglePixel(imageData, dataIndex, colors,
                     (screenDataItem >> mask) & 0xff);
      // Build the image from top to bottom, left to right
      if (y >= height - 1) { y = 0; x += 4; } else { y += 1; }
    }
  }
  context.putImageData(imageData, 0, 0);
};
function setSinglePixel(imageData, dataIndex, colors, colorIndex) {
  const color = BINDING.mono_array_get(colors, colorIndex);
  imageData.data[dataIndex] = color & 0xff;
  imageData.data[dataIndex + 1] = (color >> 8) & 0xff;
  imageData.data[dataIndex + 2] = (color >> 16) & 0xff;
  imageData.data[dataIndex + 3] = 255;
}
```

Tips and lessons learned

- Avoid Array.Copy on Big arrays (in .Net 5)
- Extensive logging from Blazor slooooows the app
- Calling Blazor from JS is very fast
  - But has problems with certain data types
  - Undocumented APIs removed in .Net 7 in favor of JS Interop

Tips and lessons learned

- Avoid Array.Copy on Big arrays (in .Net 5)
- Extensive logging from Blazor slooooows the app
- Calling Blazor from JS is very fast
  - But has problems with certain data types
  - Undocumented APIs removed in .Net 7 in favor of JS Interop

- window.requestAnimationFrame allows to pace the frames
- Browsers require interaction with the page to play audio

DEMO

JS Interop in .net >= 7

- Less intricate way to run .Net from JS (no components)
- More adapted to this case than Blazor

Call JS from .Net

```js
export function setLocalStorage(todosJson) {
  window.localStorage.setItem('dotnet-wasm-todomvc', todosJson);
}
```

JS

```cs
static partial class Interop
{
    [JSImport("setLocalStorage", "todoMVC/store.js")]
    internal static partial void _setLocalStorage(string json);
}
```

C#

Call .Net from JS

```cs
public partial class MainJS
{
    [JSExport]
    public static void OnHashchange(string url)
    {
        controller?.SetView(url);
    }
}
```

C#

```js
const exports = await getAssemblyExports(
  getConfig().mainAssemblyName);
exports.TodoMVC.MainJS.OnHashchange(document.location.hash);
```

JS

Next steps

- Short term:
  - Migrate to JS Interop
  - Update to ManagedDoom V2
- Middle term:
  - Game music
  - Test WADs othen than DOOM1
- Long term / wish:
  - PR this port to ManagedDoom

Conclusion

- WASM makes existing code compatible with Browsers
- Porting games is fun

Source code

Slides

Thanks !

Any questions ?

Links

- https://mspoweruser.com/this-doom-digital-camera-source-port-is-amazing-and-bizarre/
- https://www.link-cable.com/top-10-weird-doom-ports/
- pixabay.com