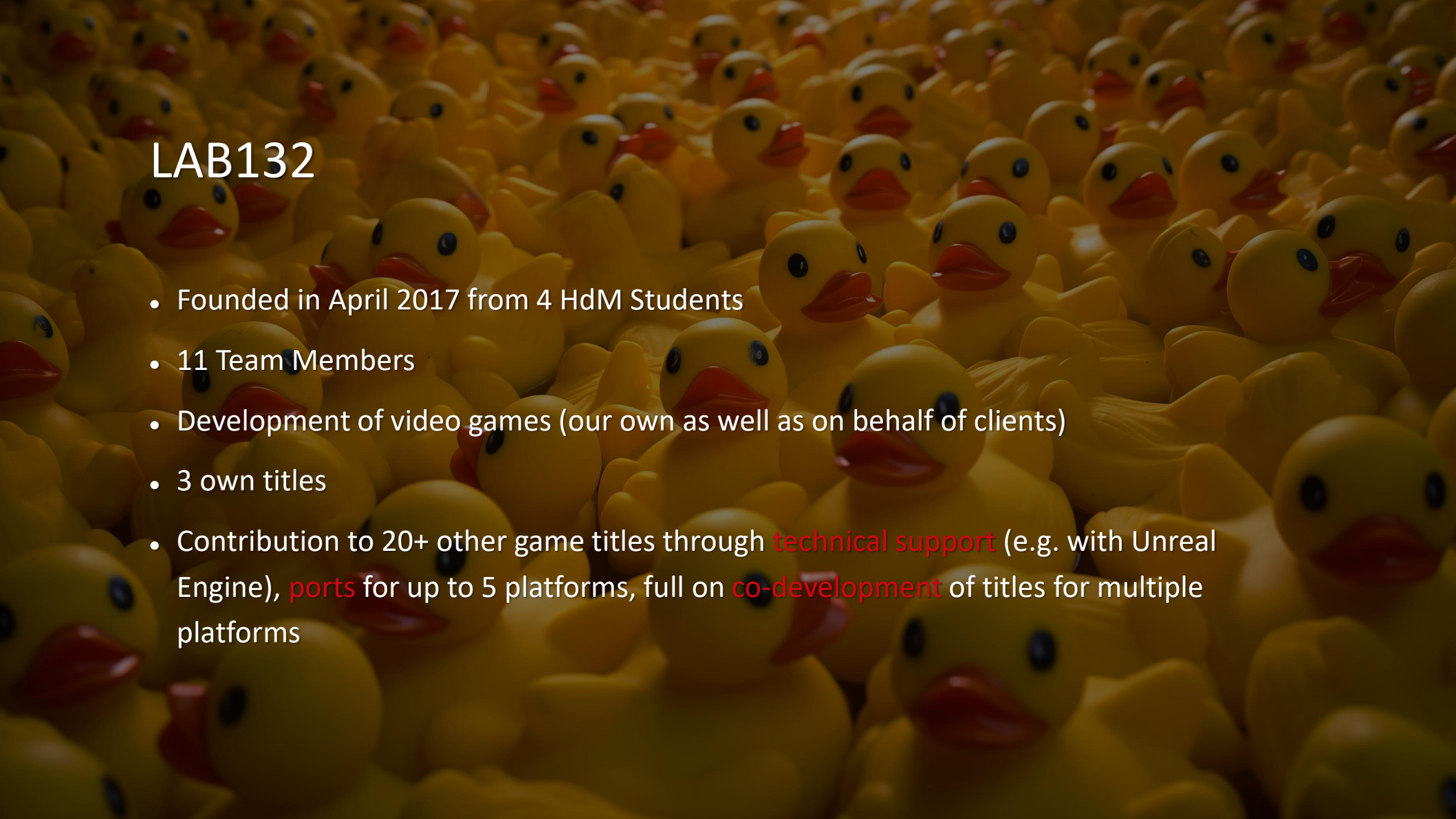
PRAGMATIC (GAME) DEVELOPMENT

MARVIN POHL

LAB 132



LAB132

- Founded in April 2017 from 4 HdM Students
- 11 Team Members
- Development of video games (our own as well as on behalf of clients)
- 3 own titles
- Contribution to 20+ other game titles through technical support (e.g. with Unreal Engine), ports for up to 5 platforms, full on co-development of titles for multiple platforms

GAMES

And more...

BACKGROUND

Best practices from 10+ years of active C++ development
and 8+ years of Unreal Engine, starting from 4.5

Working on different codebases from different teams,
reviewing code from employees.
Implementing features in a timely and stable manner.

Samples are from Unreal Engine,
but concepts apply to all kinds of development.

PRAGMATIC DEVELOPMENT

Not just getting the job done

Feature separation

Ensuring maintainability

00. MAINTAINABILITY

- 80% of the lifetime cost of a piece of software goes to maintenance.
  - Maintenance – aka Refactor, Polishing, Debugging, Reading, …
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of software, allowing engineers to understand new code quickly and thoroughly.

https://docs.unrealengine.com/5.3/en-US/epic-cplusplus-coding-standard-for-unreal-engine/

01. CODE STYLE MATTERS MORE THAN YOU THINK

- Makes it readable
- Avoids common errors
- Do not leave commented out code

01. CODE STYLE MATTERS MORE THAN YOU THINK

```
const float rps = m_erpm / 60.0f;
const float v =  (m_parent->GetPhysicsLinearVelocityAtPoint(GetComponentLocation()) | -GetForwardVector());
const float av = FMath::Abs(v);
float s = m_prop->getFWS(m_prop->m_maxrpm);
if (!ensure(!FMath::IsNearlyZero(s)))
{
    s = 1.0f;
}
const float r = av / s;
const float c = v < 0.0f ? m_prop->m_coeff_rev * 0.01f : 1.0f;
const float fc = m_prop->m_coeff.GetRichCurveConst()->Eval(InTime:r);
const float d = m_prop->m_d * m_prop->m_d * 0.0001f;
const float f = FMath::Sign(rps) * fc * m_p * rps * rps * d * d * c;
return f;
```

01. CODE STYLE MATTERS MORE THAN YOU THINK

```
const float RevolutionsPerSecond = EngineRPM / 60.0f;
const float LinearVelocity =
    (PrimitiveAttachParent->GetPhysicsLinearVelocityAtPoint(GetComponentLocation()) | -GetForwardVector());
const float AbsoluteLinearVelocity = FMath::Abs(LinearVelocity);
float FreewheelSpeed = Propeller->GetFreewheelSpeed(Propeller->MaxRPM);
// This can only happen in an incorrect setup from the propeller (either no Pitch or 0 Max RPM)
if (!ensure(!FMath::IsNearlyZero(FreewheelSpeed)))
{
    FreewheelSpeed = 1.0f;
}
const float NormalizedAdvanceRatio = AbsoluteLinearVelocity / FreewheelSpeed;

const float ReverseCoefficient = LinearVelocity < 0.0f ? Propeller->ReverseEfficiency * 0.01f : 1.0f;

const float ForceCoefficient = Propeller->ForceCoefficient.GetRichCurveConst()->Eval(InTime:NormalizedAdvanceRatio);

// cm^2->m^2
const float DiameterSquared = Propeller->Diameter * Propeller->Diameter * 0.0001f;

// F = K_F * p * n^2 * D^4
const float Force = FMath::Sign(RevolutionsPerSecond) * ForceCoefficient * WaterDensity * RevolutionsPerSecond *
                    RevolutionsPerSecond * DiameterSquared * DiameterSquared * ReverseCoefficient;

return Force;
```

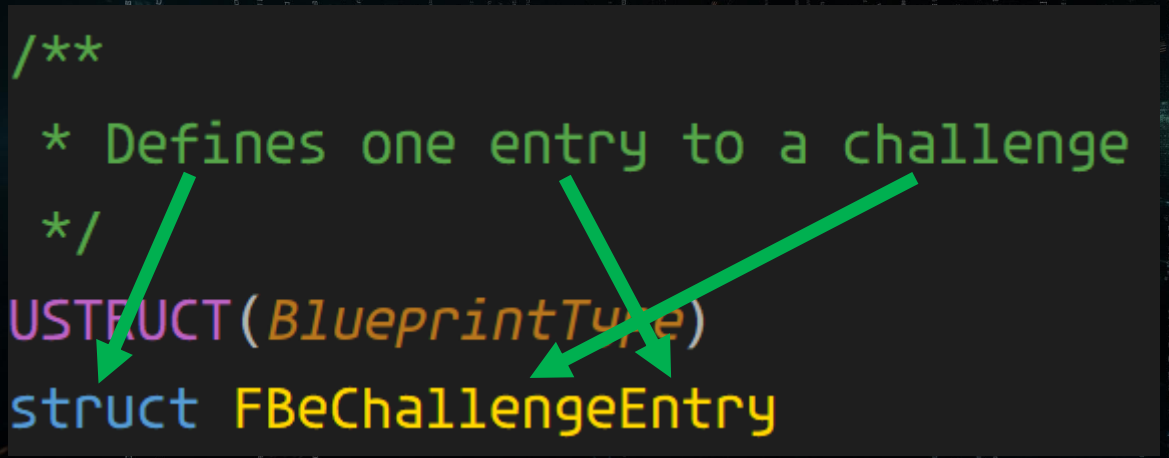
02. COMMENTS SHOULD SUPPLEMENT MISSING INFORMATION

- Self explanatory code does not need comments
- Describe unusual behavior, that is not obvious
- Edge cases and expectations to the callee or reader
- Additional information not conveyed by function/variable/class names

02. COMMENTS SHOULD SUPPLEMENT MISSING INFORMATION

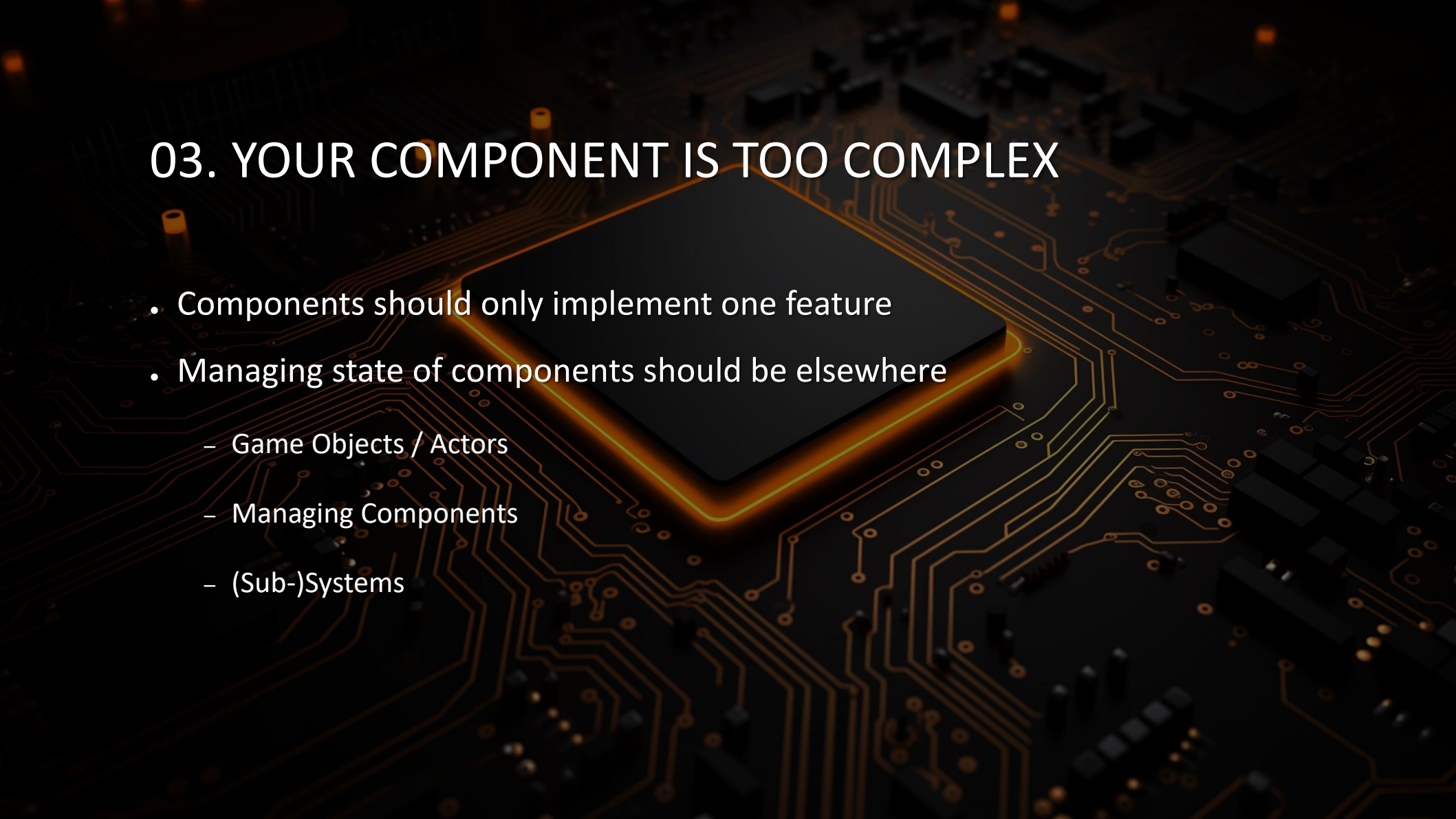
```cpp
#if WITH_EDITOR
void ABoat::RerunConstructionScripts()
{
    // Note(marvin@lab132.com): This is used by the boat balancing tool.
    // If we allow rerunning the construction script, this also recreates the (Blueprint) generated components
    // and we loose the connection to our edited properties.
    // There might be better options here,
    // but finding the correct event to replace the edited object with the reconstructed one can be tricky,
    // so this is the best solution for now.
    //
    // AdditionalNote(marvin@lab132.com): Might be also an option to implement this into the engine
    // to not rerun the construction script on a variable change.
    // This might fix some breaking actors when editing values while playing.
    // Since while we are playing and changing values in the editor,
    // one would not expect the construction script to be run anyway
    if (bAllowReconstruction)
    {
        Super::RerunConstructionScripts();
    }
}
#endif  #if WITH_EDITOR
```

02. SELF DESCRIBING COMMENT



02. BETTER EXAMPLE

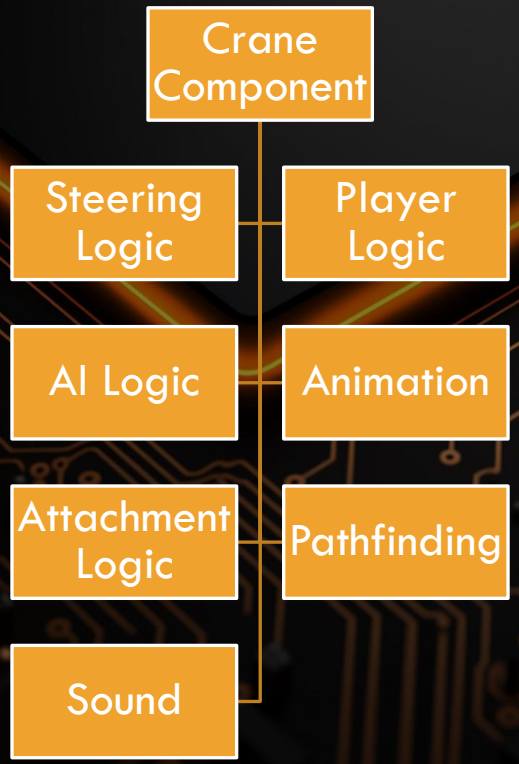
```
/**
 * The LevelRuleset defines all the rules for a level.
 * It contains the definition on how to generate the path and how to decorate it.
 * Combined with a PathSeed, it can be used to generate a level and serves as a unique identifier for a level.
 * More specific, the ruleset's primary asset id is used as the unique identifier part.
 */
UCLASS()
No derived blueprint classes
class BERYLLIUM_API UBeLevelRuleset : public UPrimaryDataAsset
```



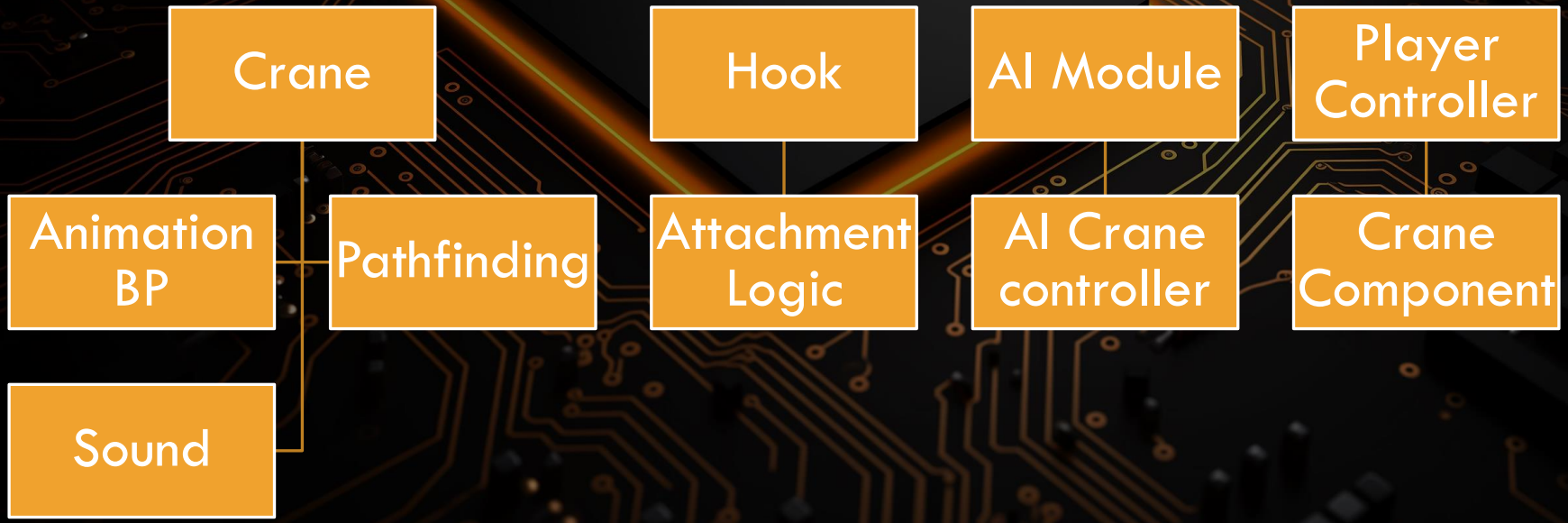
03. YOUR COMPONENT IS TOO COMPLEX

- Components should only implement one feature
- Managing state of components should be elsewhere
  - Game Objects / Actors
  - Managing Components
  - (Sub-)Systems

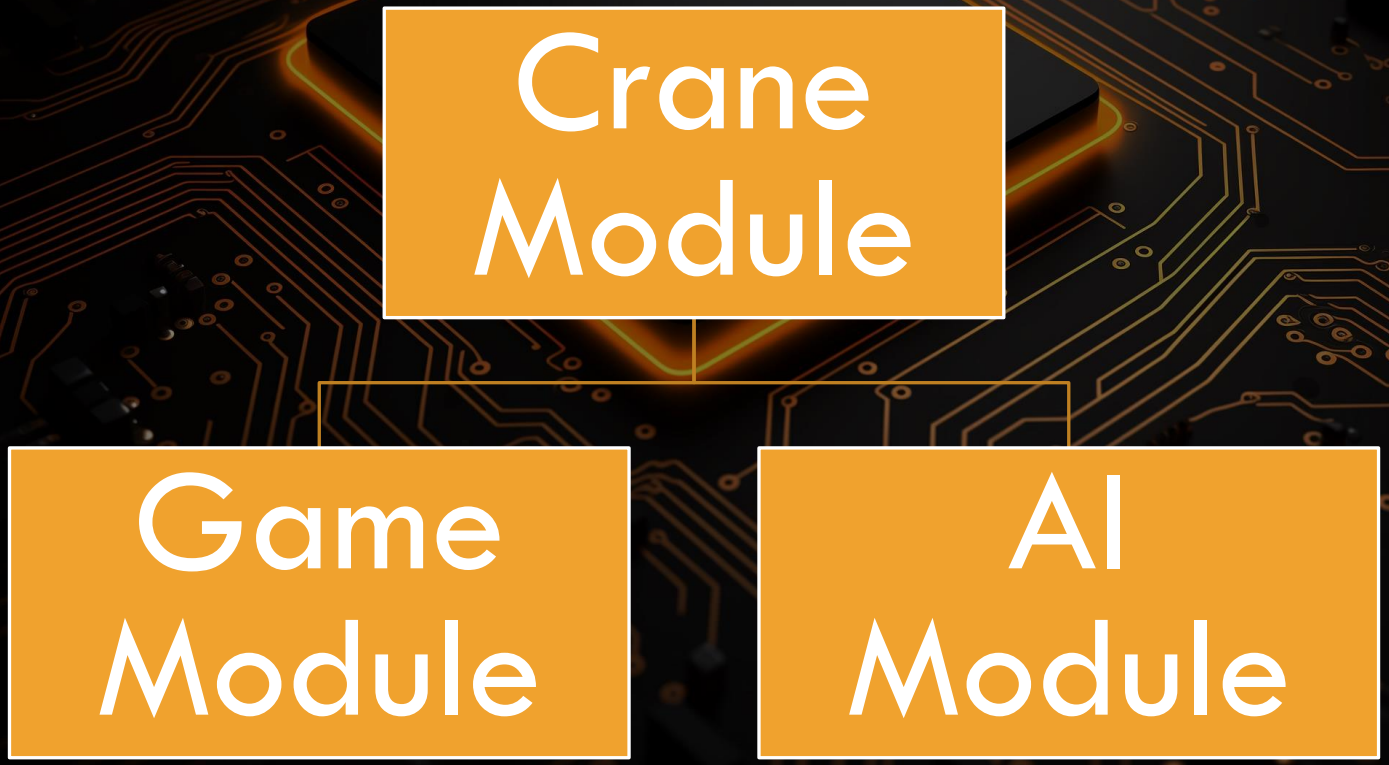
03. YOUR COMPONENT IS TOO COMPLEX



03. YOUR COMPONENT IS TOO COMPLEX

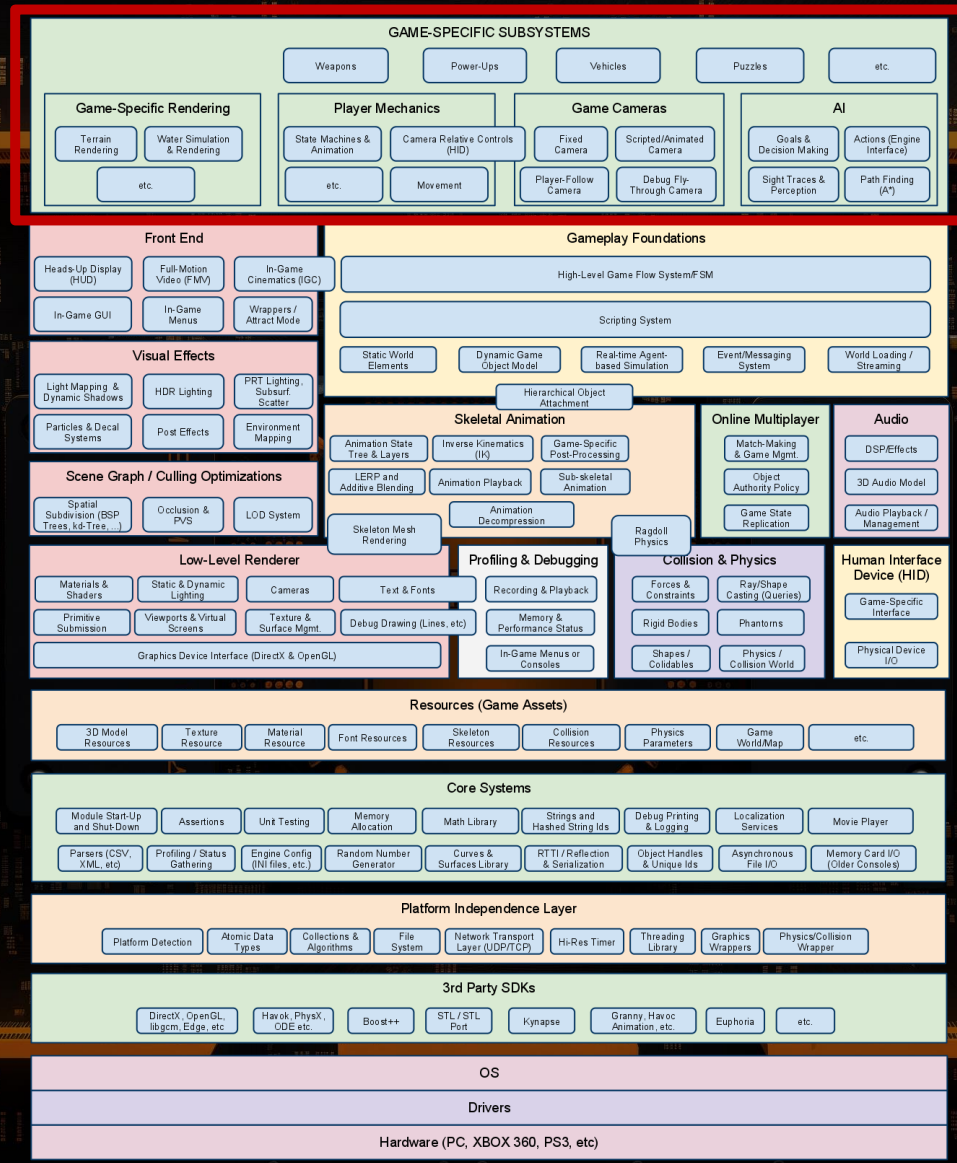


03. YOUR COMPONENT IS TOO COMPLEX

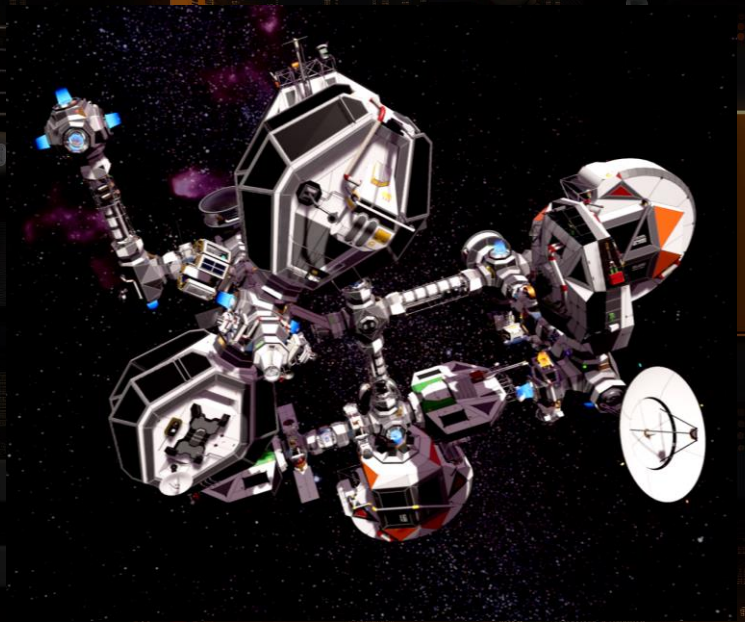


04. YOUR FEATURE SHOULD BE A (SUB-)SYSTEM

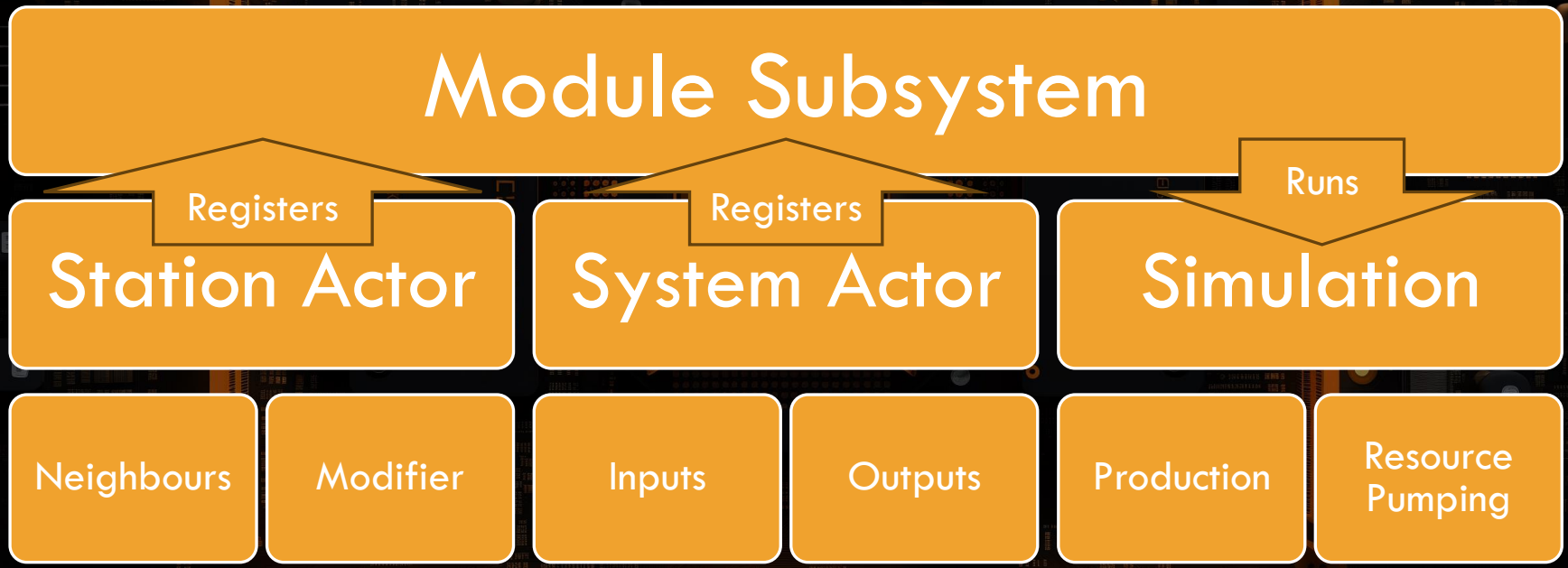
- Systemic Design is not restricted to the engine
- Separate managing part of features into systems
- Helps avoid state bugs when used by multiple sources


GAME-SPECIFIC SUBSYSTEMS
Weapons
Power-Ups
Vehicles
Puzzles
etc.
Game-Specific Rendering
Terrain Rendering
Water Simulation & Rendering
etc.
Player Mechanics
State Machines & Animation
Camera Relative Controls (HID)
etc.
Movement
Game Cameras
Fixed Camera
Scripted/Animated Camera
Player-Follow Camera
Debug Fly-Through Camera
AI
Goals & Decision Making
Actions (Engine Interface)
Sight Traces & Perception
Path Finding (A*)
Front End
Heads-Up Display (HUD)
Full-Motion Video (FMV)
In-Game Cinematics (IGC)
In-Game GUI
In-Game Menus
Wrappers / Attract Mode
Visual Effects
Light Mapping & Dynamic Shadows
HDR Lighting
PRT Lighting, Subsurf. Scatter
Particles & Decal Systems
Post Effects
Environment Mapping
Scene Graph / Culling Optimizations
Spatial Subdivision (BSP Trees, kd-Tree,...)
Occlusion & PVS
LOD System
Gameplay Foundations
High-Level Game Flow System/FSM
Scripting System
Static World Elements
Dynamic Game Object Model
Real-time Agent-based Simulation
Event/Messaging System
World Loading / Streaming
Hierarchical Object Attachment
Skeletal Animation
Animation State Tree & Layers
Inverse Kinematics (IK)
Game-Specific Post-Processing
LERP and Additive Blending
Animation Playback
Sub-skeletal Animation
Animation Decompression
Skeleton Mesh Rendering
Ragdoll Physics
Online Multiplayer
Match-Making & Game Mgmt.
Object Authority Policy
Game State Replication
Audio
DSP/Effects
3D Audio Model
Audio Playback / Management
Low-Level Renderer
Materials & Shaders
Static & Dynamic Lighting
Cameras
Text & Fonts
Primitive Submission
Viewports & Virtual Screens
Texture & Surface Mgmt.
Debug Drawing (Lines, etc)
Graphics Device Interface (DirectX & OpenGL)
Profiling & Debugging
Recording & Playback
Memory & Performance Status
In-Game Menus or Consoles
Collision & Physics
Forces & Constraints
Ray/Shape Casting (Queries)
Rigid Bodies
Phantoms
Shapes / Collidables
Physics / Collision World
Human Interface Device (HID)
Game-Specific Interface
Physical Device I/O
Resources (Game Assets)
3D Model Resources
Texture Resource
Material Resource
Font Resources
Skeleton Resources
Collision Resources
Physics Parameters
Game World/Map
etc.
Core Systems
Module Start-Up and Shut-Down
Assertions
Unit Testing
Memory Allocation
Math Library
Strings and Hashed String Ids
Debug Printing & Logging
Localization Services
Movie Player
Parsers (CSV, XML, etc)
Profiling / Status Gathering
Engine Config (INI files, etc.)
Random Number Generator
Curves & Surfaces Library
RTTI / Reflection & Serialization
Object Handles & Unique Ids
Asynchronous File I/O
Memory Card I/O (Older Consoles)
Platform Independence Layer
Platform Detection
Atomic Data Types
Collections & Algorithms
File System
Network Transport Layer (UDP/TCP)
Hi-Res Timer
Threading Library
Graphics Wrappers
Physics/Collision Wrapper
3rd Party SDKs
DirectX, OpenGL, libgcm, Edge, etc.
Havok, PhysX, ODE etc.
Boost++
STL / STL Port
Kynapse
Granny, Havoc Animation, etc.
Euphoria
etc.
OS
Drivers
Hardware (PC, XBOX 360, PS3, etc)

04. YOUR FEATURE SHOULD BE A (SUB-)SYSTEM

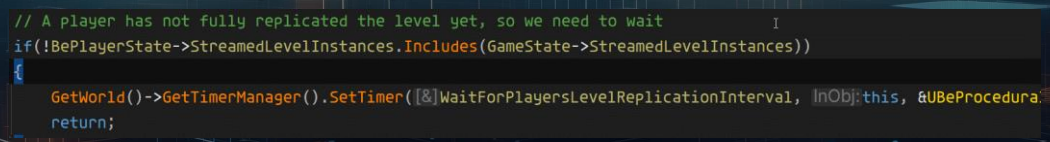


04. YOUR FEATURE SHOULD BE A (SUB-)SYSTEM



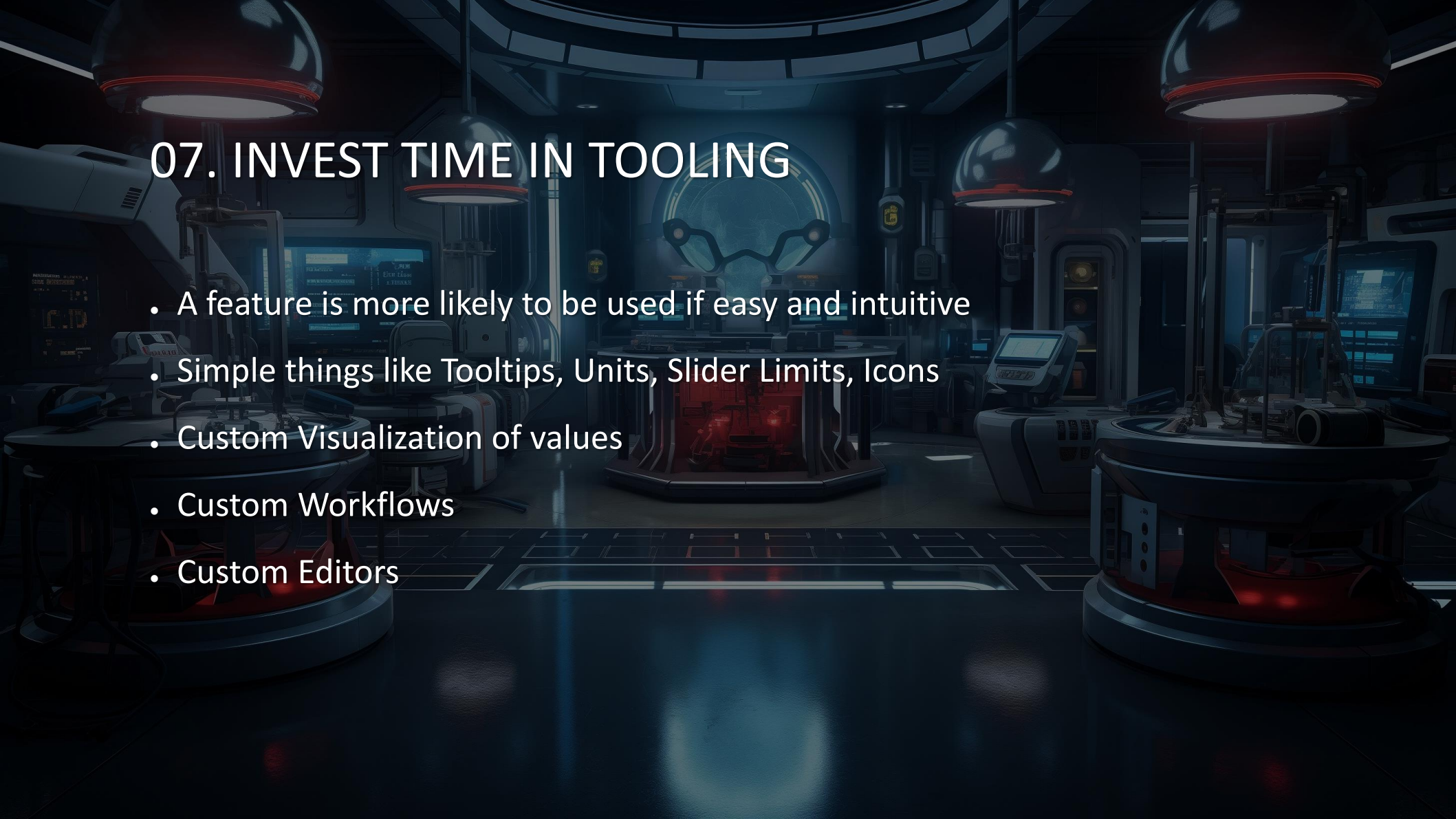
06. DON’T THROW AWAY YOUR DEBUG VISUALIZATION

- Make it accessible via debug flags
- Keep your debug logs (or write them in the first place)
  - (GitHub) Copilot is your friend 👀



- Do not comment out debug code
- Avoid conditional compilation if performance allows it

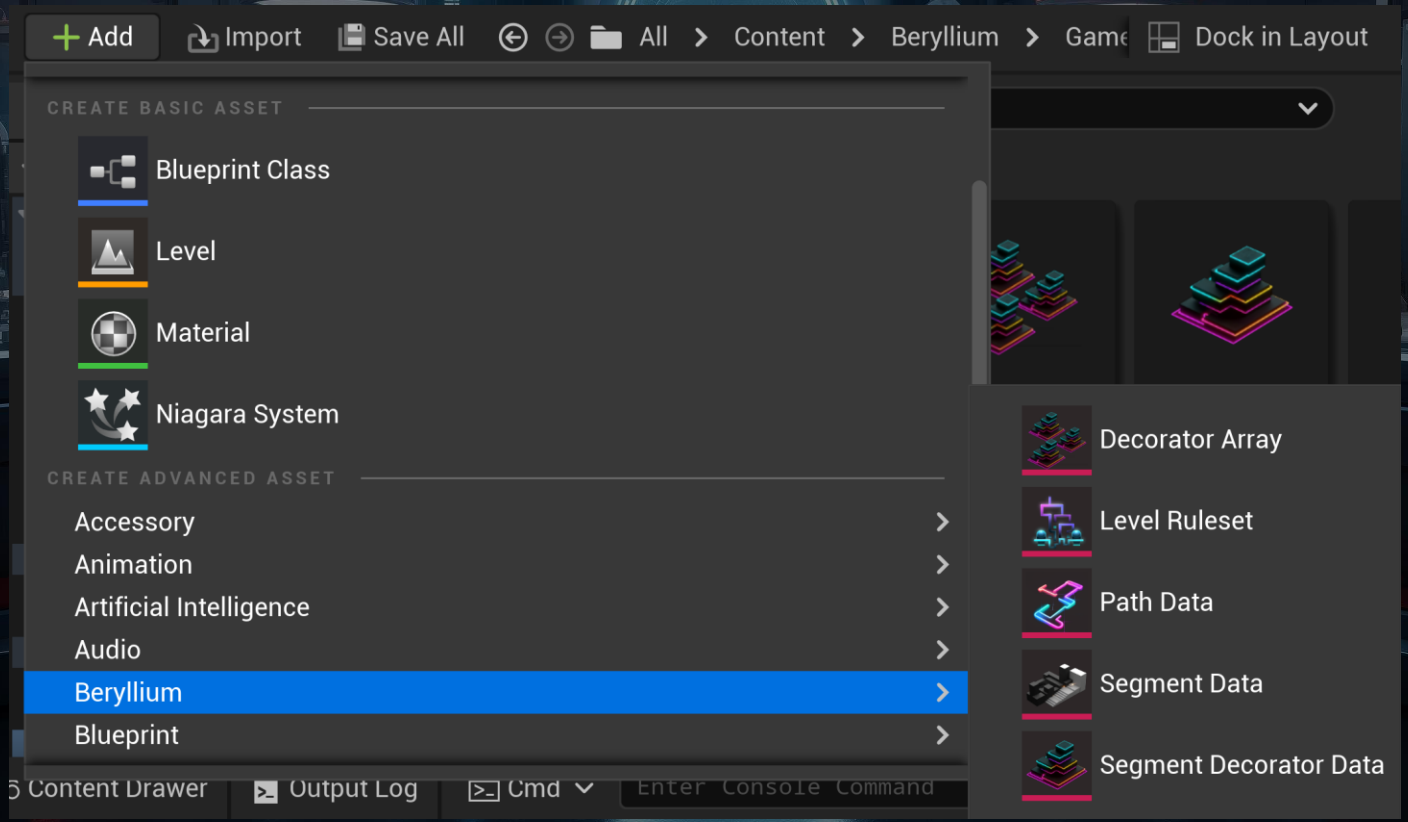
06. DON’T THROW AWAY YOUR DEBUG VISUALIZATION



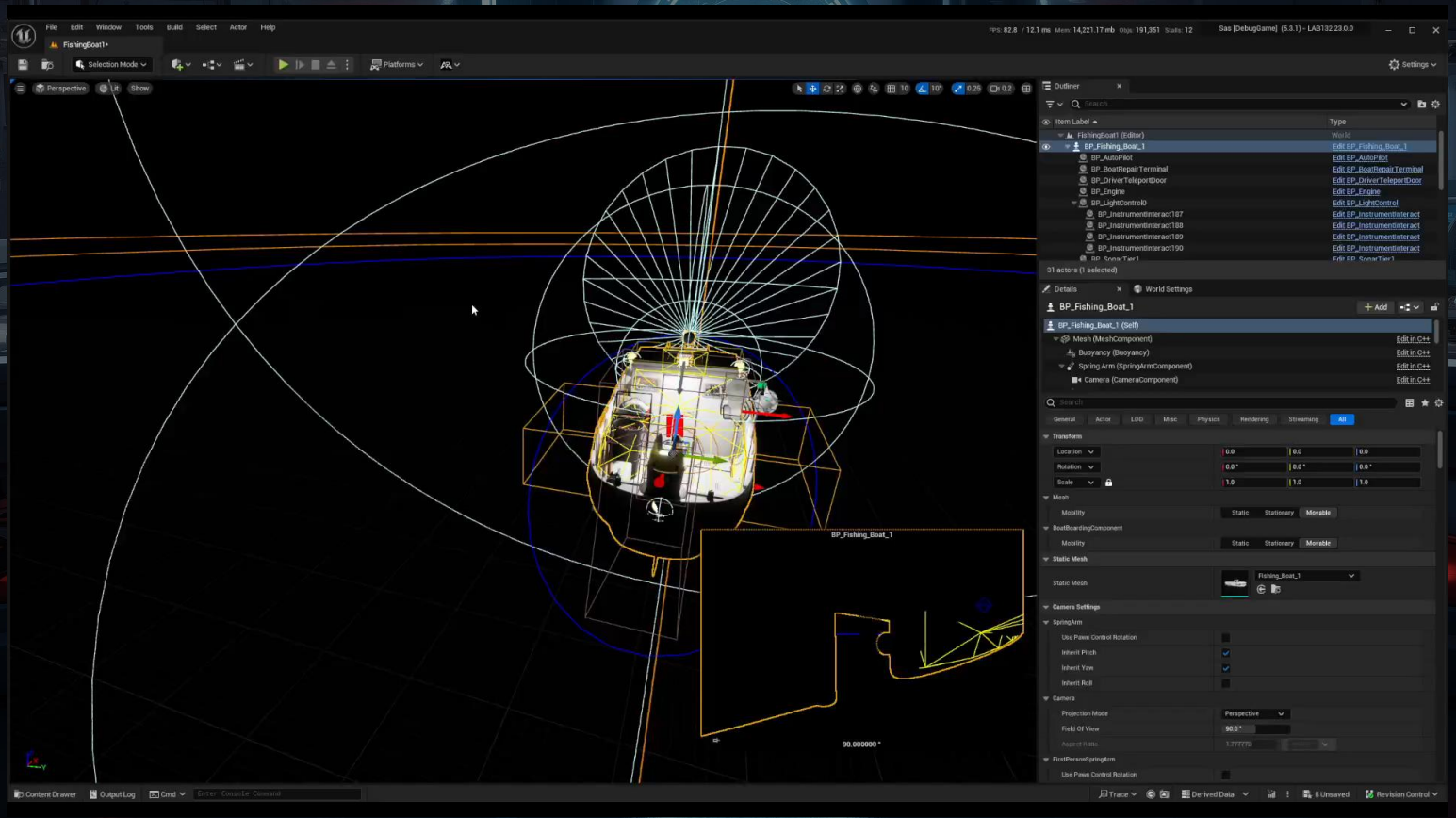
07. INVEST TIME IN TOOLING

- A feature is more likely to be used if easy and intuitive
- Simple things like Tooltips, Units, Slider Limits, Icons
- Custom Visualization of values
- Custom Workflows
- Custom Editors

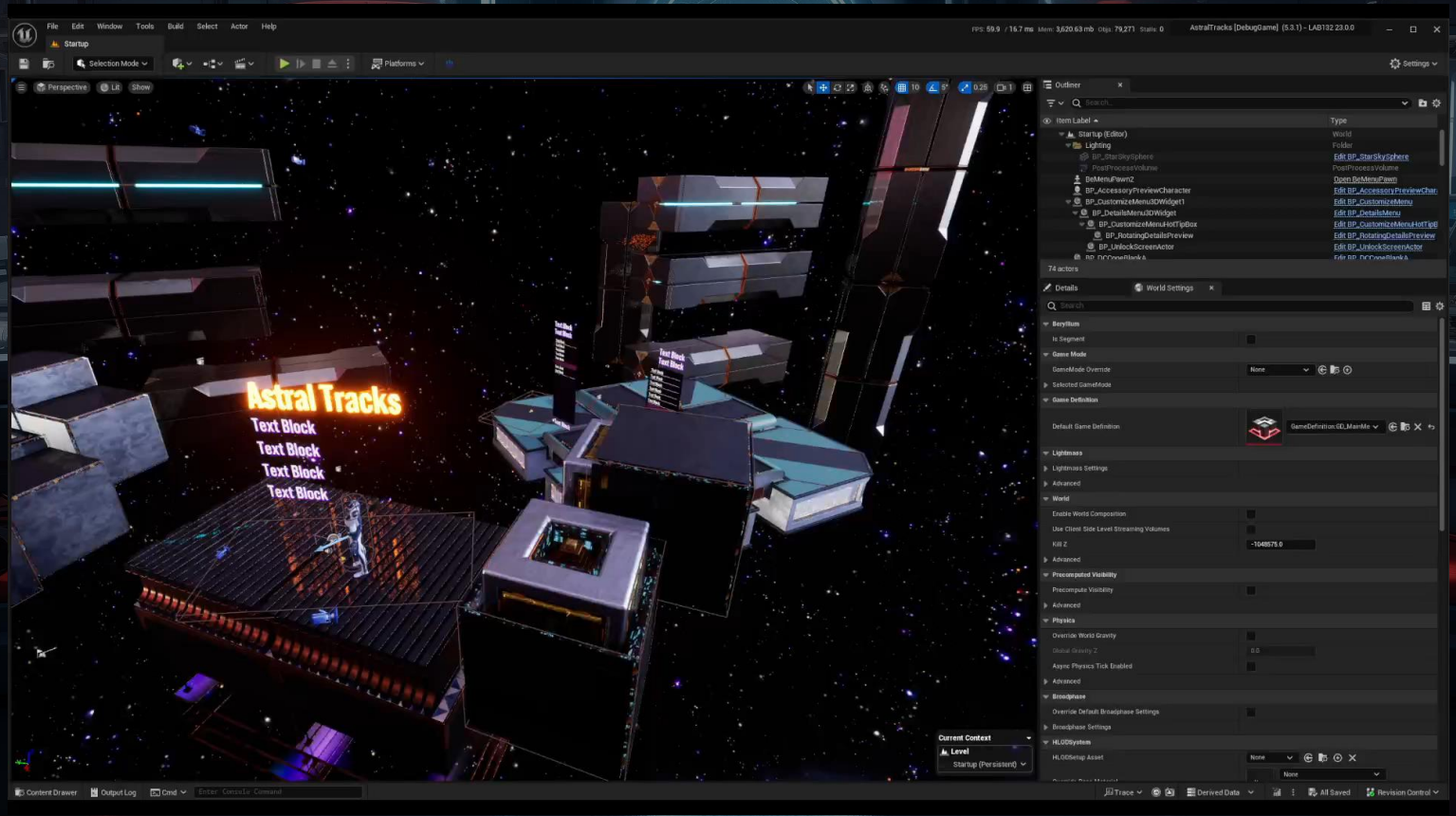
07. INVEST TIME IN TOOLING



07. INVEST TIME IN TOOLING



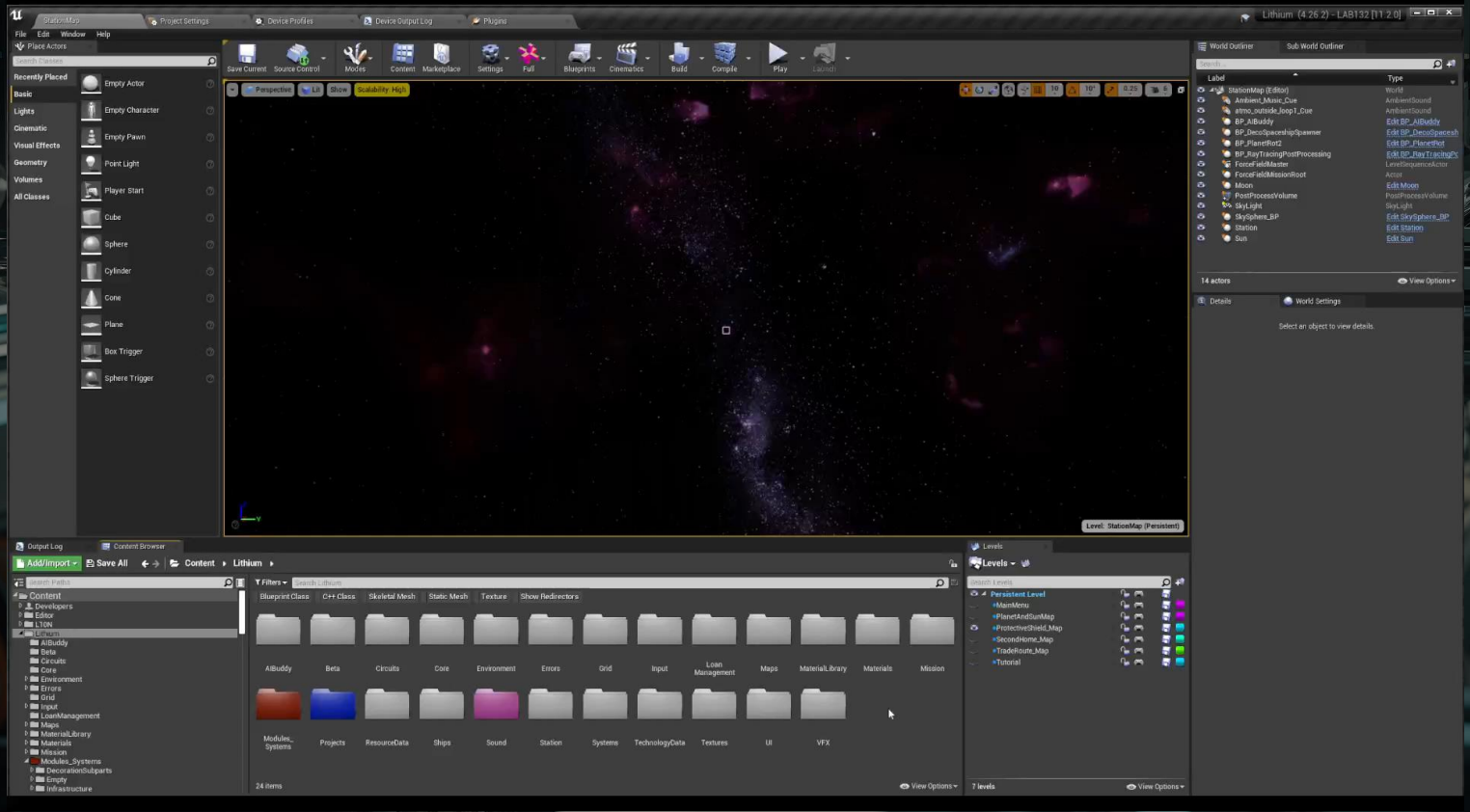
07. INVEST TIME IN TOOLING



08. AUTOMATE YOUR WORKFLOW

- No one likes busywork
- Prevents copy  or careless errors
- Asset creation
- Validation

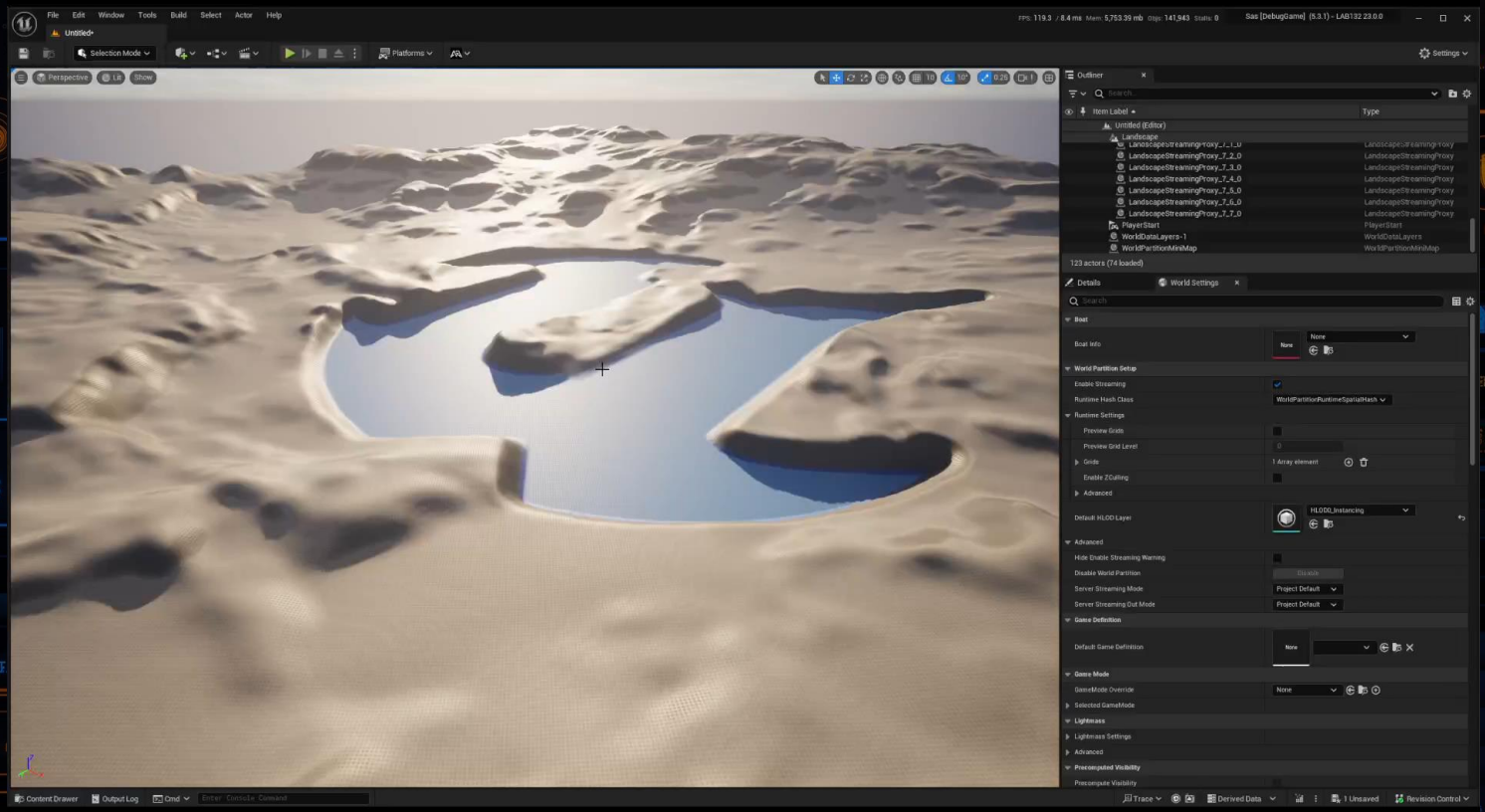
08. AUTOMATE YOUR WORKFLOW



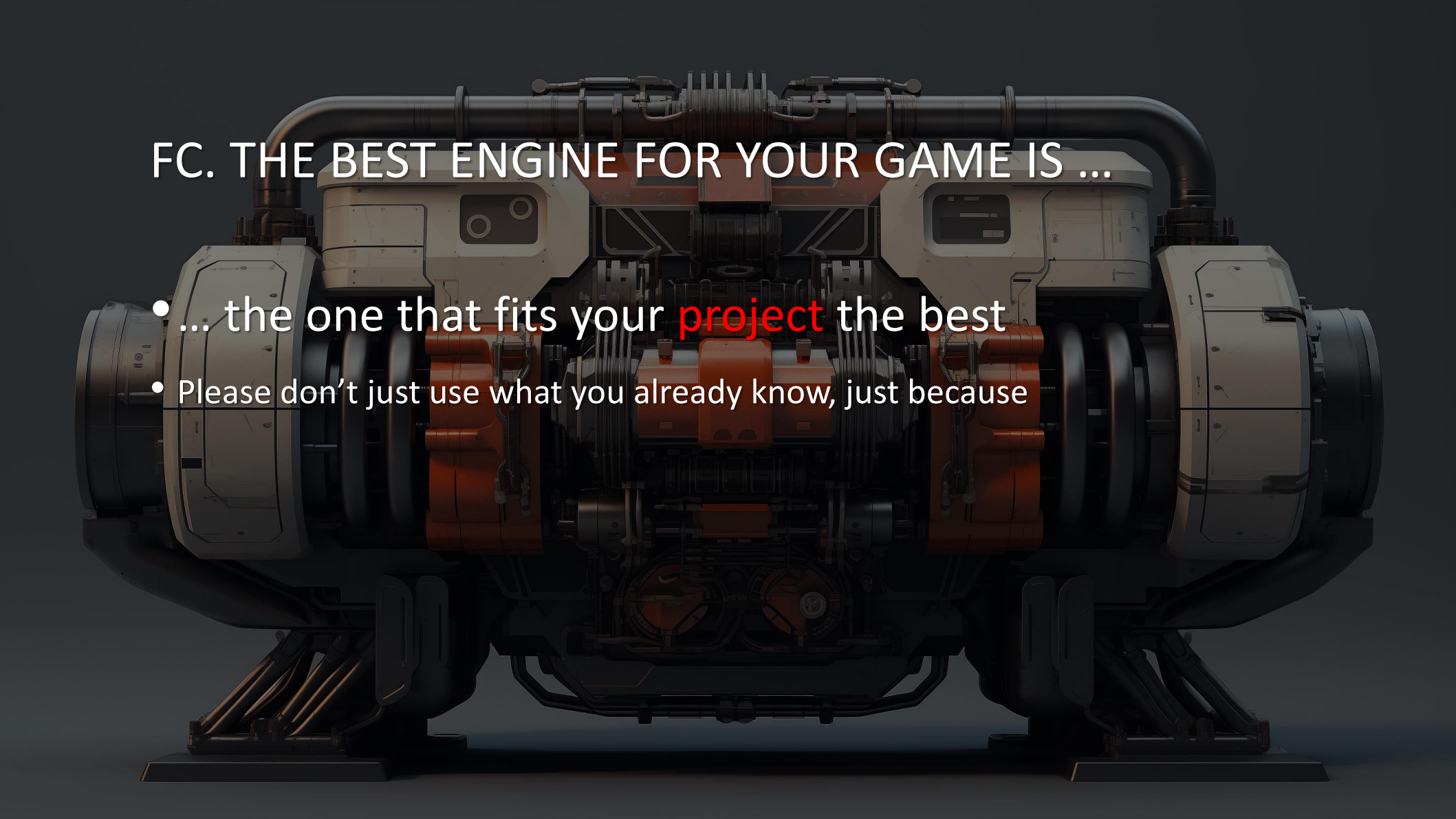
09. YOUR DEFAULT VALUE SHOULD NOT BE 0

- Your feature/component should work out of the box
- Do not hardcode values
- Gracefully handle broken user facing values
  - An editor crash can mean hours of wasted time because someone forgot to save

09. YOUR DEFAULT VALUE SHOULD NOT BE 0

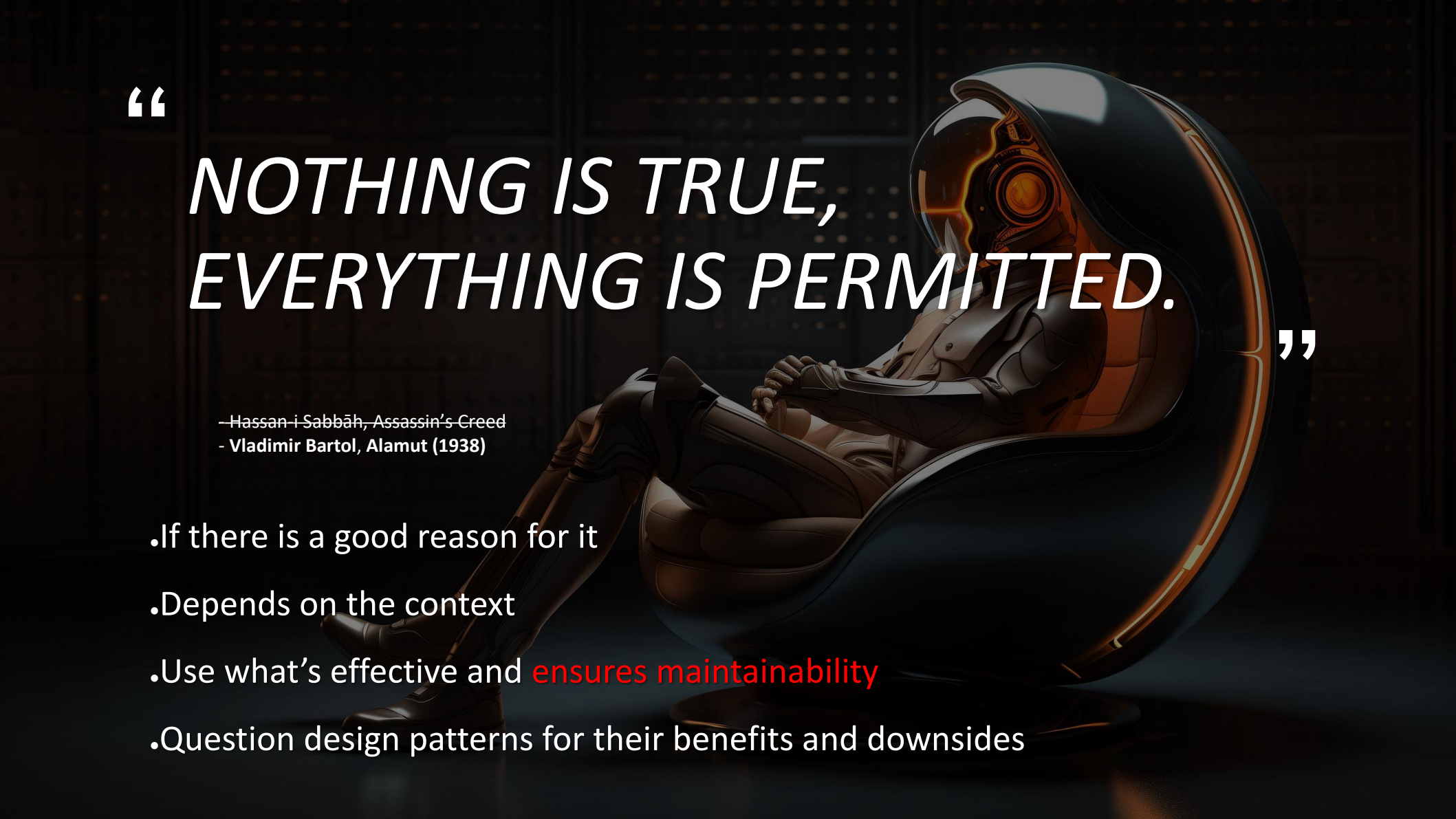


FC. THE BEST ENGINE FOR YOUR GAME IS …



FC. THE BEST ENGINE FOR YOUR GAME IS …

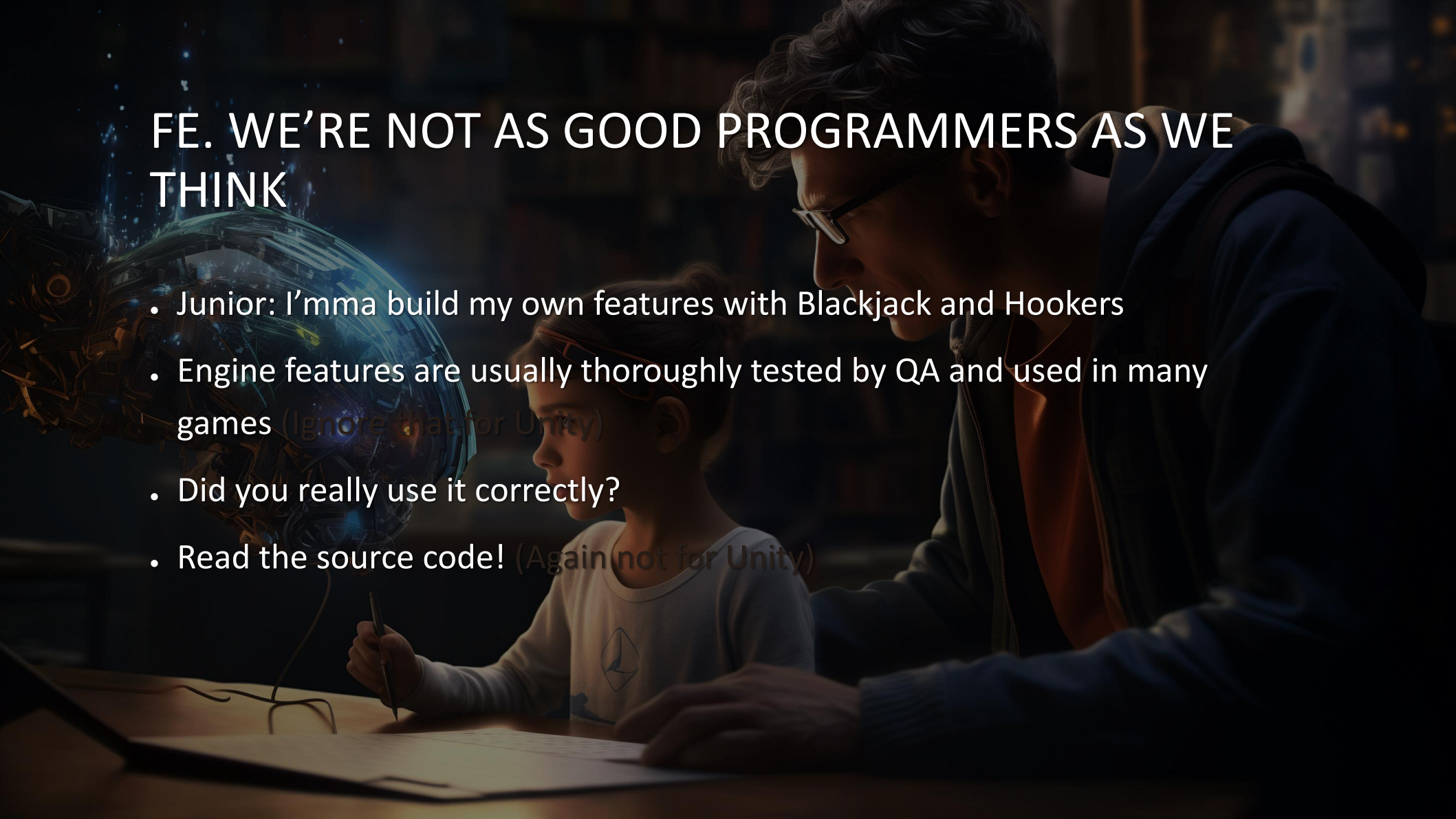
- … the one that fits your project the best
- Please don’t just use what you already know, just because



“*NOTHING IS TRUE, EVERYTHING IS PERMITTED.*”

~~- Hassan-i Sabbāh, Assassin’s Creed~~
- **Vladimir Bartol, Alamut (1938)**

- If there is a good reason for it
- Depends on the context
- Use what’s effective and ensures maintainability
- Question design patterns for their benefits and downsides



FE. WE'RE NOT AS GOOD PROGRAMMERS AS WE THINK

- Junior: I'mma build my own features with Blackjack and Hookers
- Engine features are usually thoroughly tested by QA and used in many games (Ignore that for Unity)
- Did you really use it correctly?
- Read the source code! (Again not for Unity)

FF. KNOW YOUR LEVEL OF KNOWLEDGE

~~Dunning-Kruger~~ effect


QUACKS?