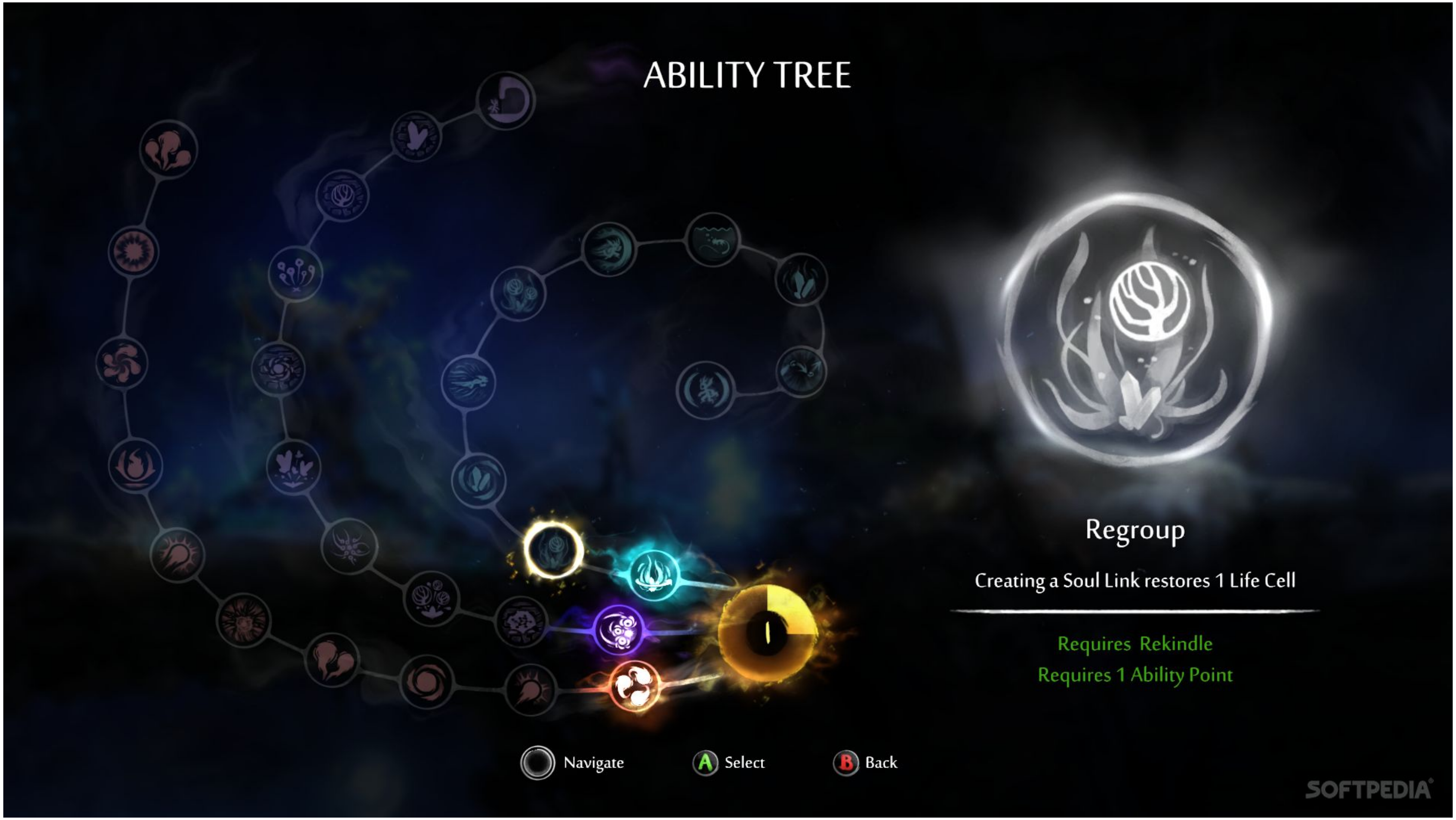
CS354P

DR SARAH ABRAHAM

OVERVIEW: GUIS

GRAPHICAL USER INTERFACES



ABILITY TREE

Regroup

Creating a Soul Link restores 1 Life Cell

Requires Rekindle
Requires 1 Ability Point

WHAT IS IN A GUI?

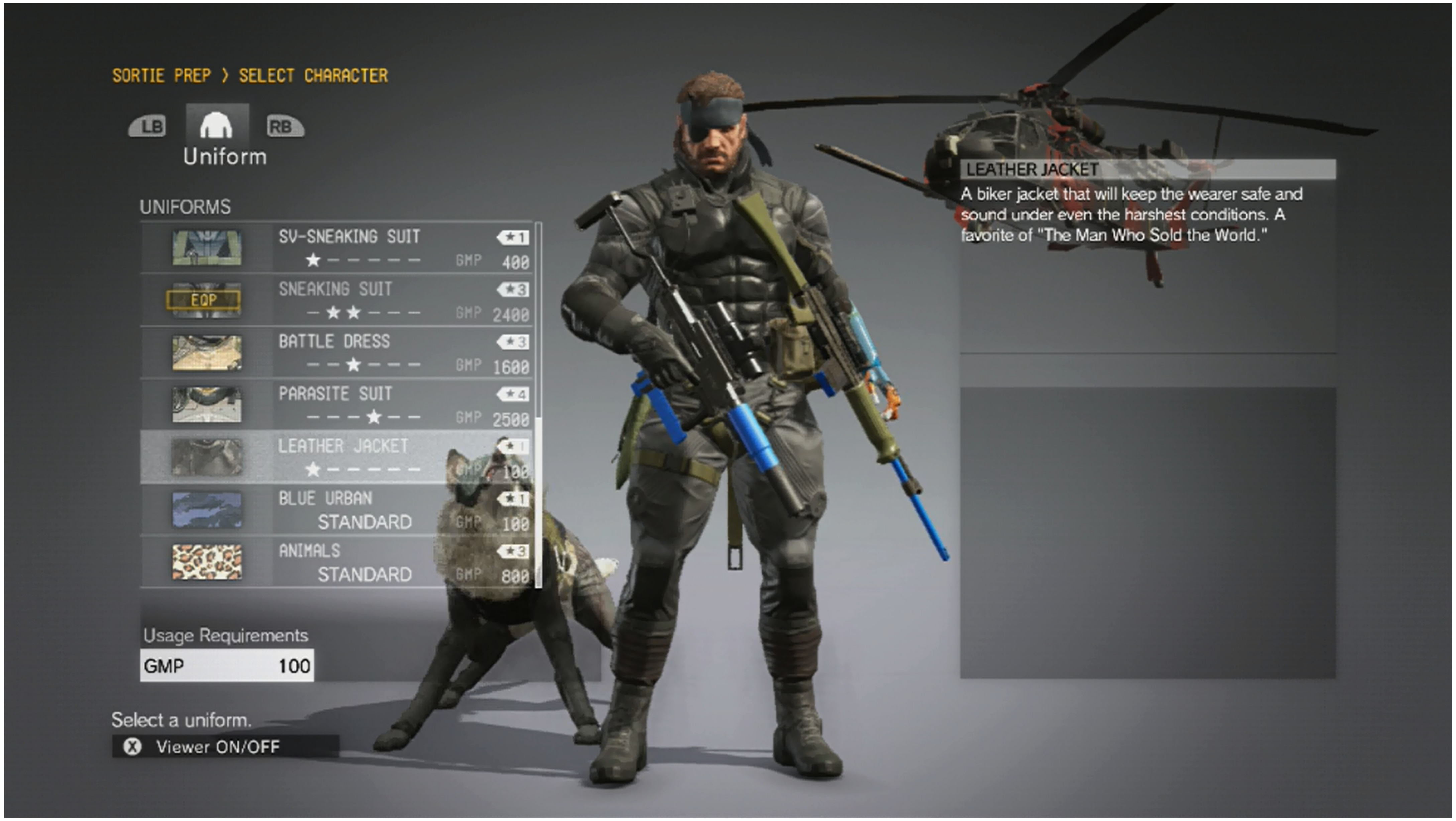
- Not just art assets!
- GUIs display important information for the player:
  - Character status
  - Enemy status
  - Leveling information
  - Map information
  - Out of game menus

DESIGNING A GUI

- GUI layouts should be:
  - Intuitive to navigate
  - Intuitive to understand
  - Intuitive to access
- This is harder than it sounds
- An entire area of design is dedicated to interaction
- You will probably get it wrong the first time
- Iterate GUI design via user testing

GUI TYPES: MENUS

- Outside of game play options, modes, and information


SORTIE PREP 〉 SELECT CHARACTER
LB
Uniform
RB
UNIFORMS
SV-SNEAKING SUIT ★1 GMP 400
EQP
SNEAKING SUIT ★3 GMP 2400
BATTLE DRESS ★3 GMP 1600
PARASITE SUIT ★4 GMP 2500
LEATHER JACKET ★1 GMP 100
BLUE URBAN STANDARD ★1 GMP 100
ANIMALS STANDARD ★3 GMP 800
Usage Requirements
GMP 100
Ⓧ Viewer ON/OFF

LEATHER JACKET

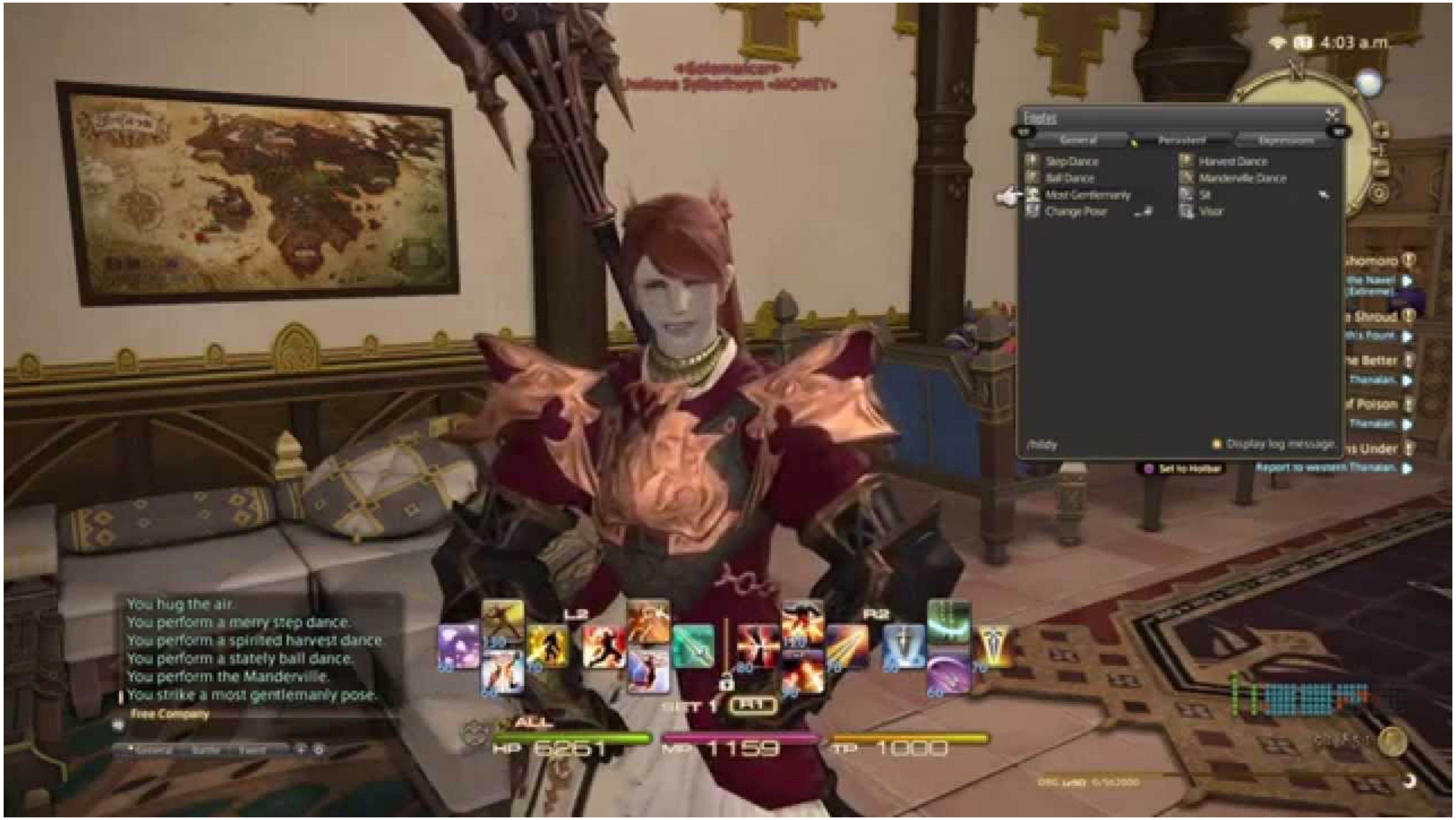
A biker jacket that will keep the wearer safe and sound under even the harshest conditions. A favorite of "The Man Who Sold the World."

Select a uniform.

Metal Gear Solid V

GUI TYPES: HUDS

- In-game persistent display of information

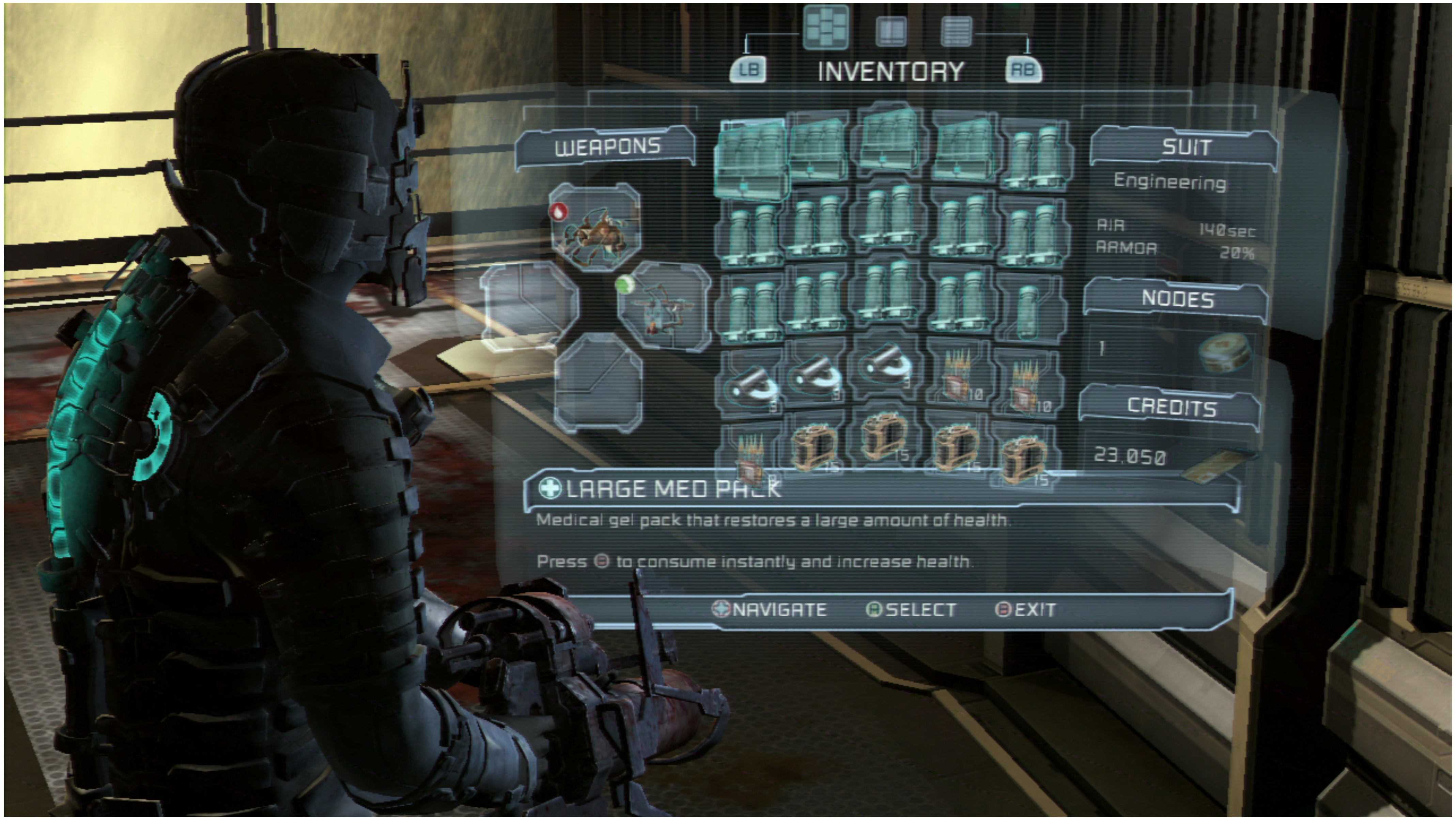


You hug the air.
You perform a merry step dance.
You perform a spirited harvest dance.
You perform a stately ball dance.
You perform the Manderville.
You strike a most gentlemanly pose.

Final Fantasy XIV

GUI TYPES: DIEGETIC DISPLAYS

- In-game display of information incorporated into world



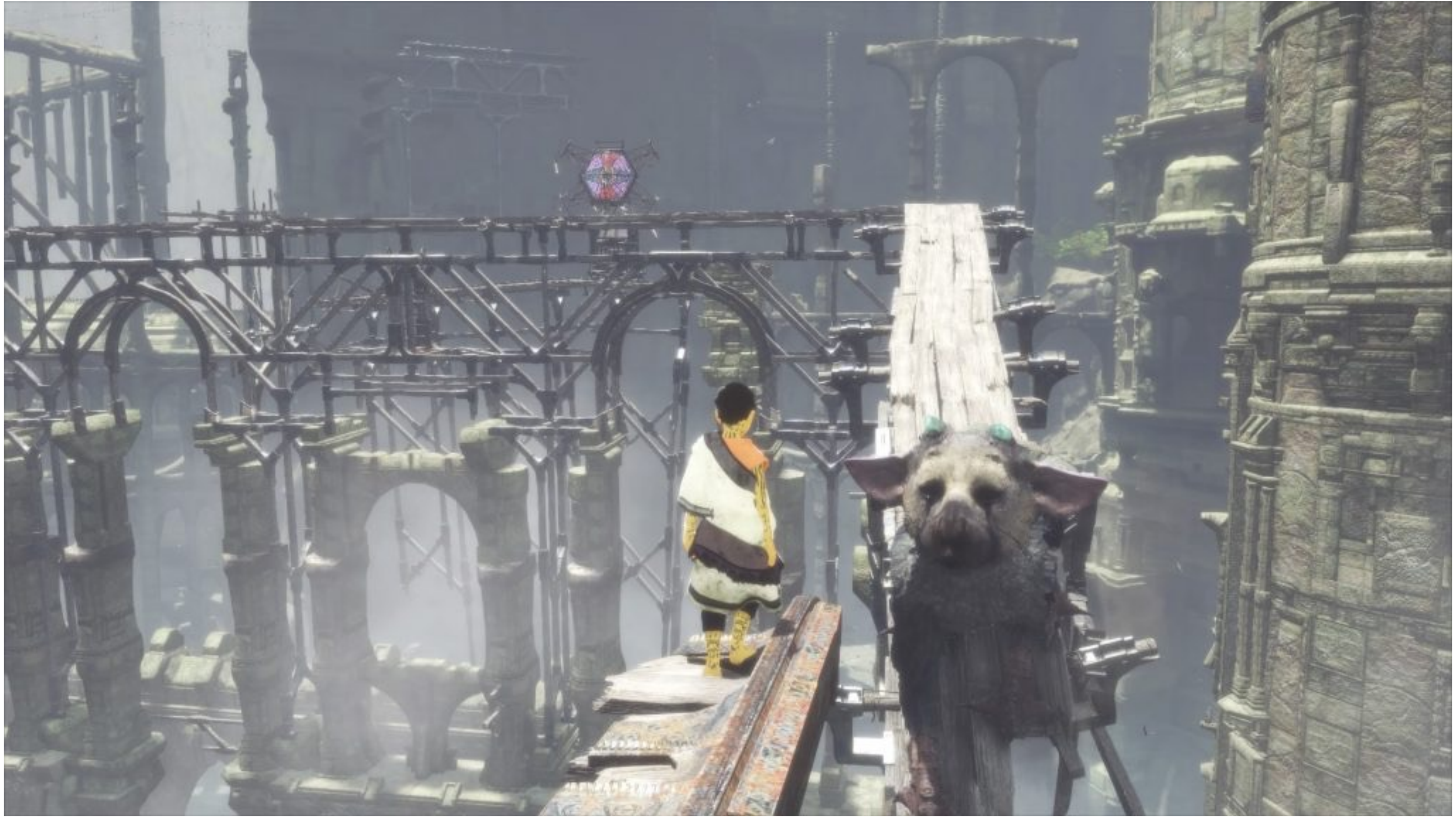
Medical gel pack that restores a large amount of health.

Press ⊜ to consume instantly and increase health.

Dead Space

GUI TYPES: GUI-LESS

- No in-game display of information – purely contextual



Last Guardian

GUI PROGRAMMING

- Based on the above, what can we determine about GUI programming?
- GUI programming is:
  - Interdisciplinary in nature
  - Highly event-driven
  - Highly state-based
  - Un-performant if implemented poorly
  - Notoriously “spaghetti”

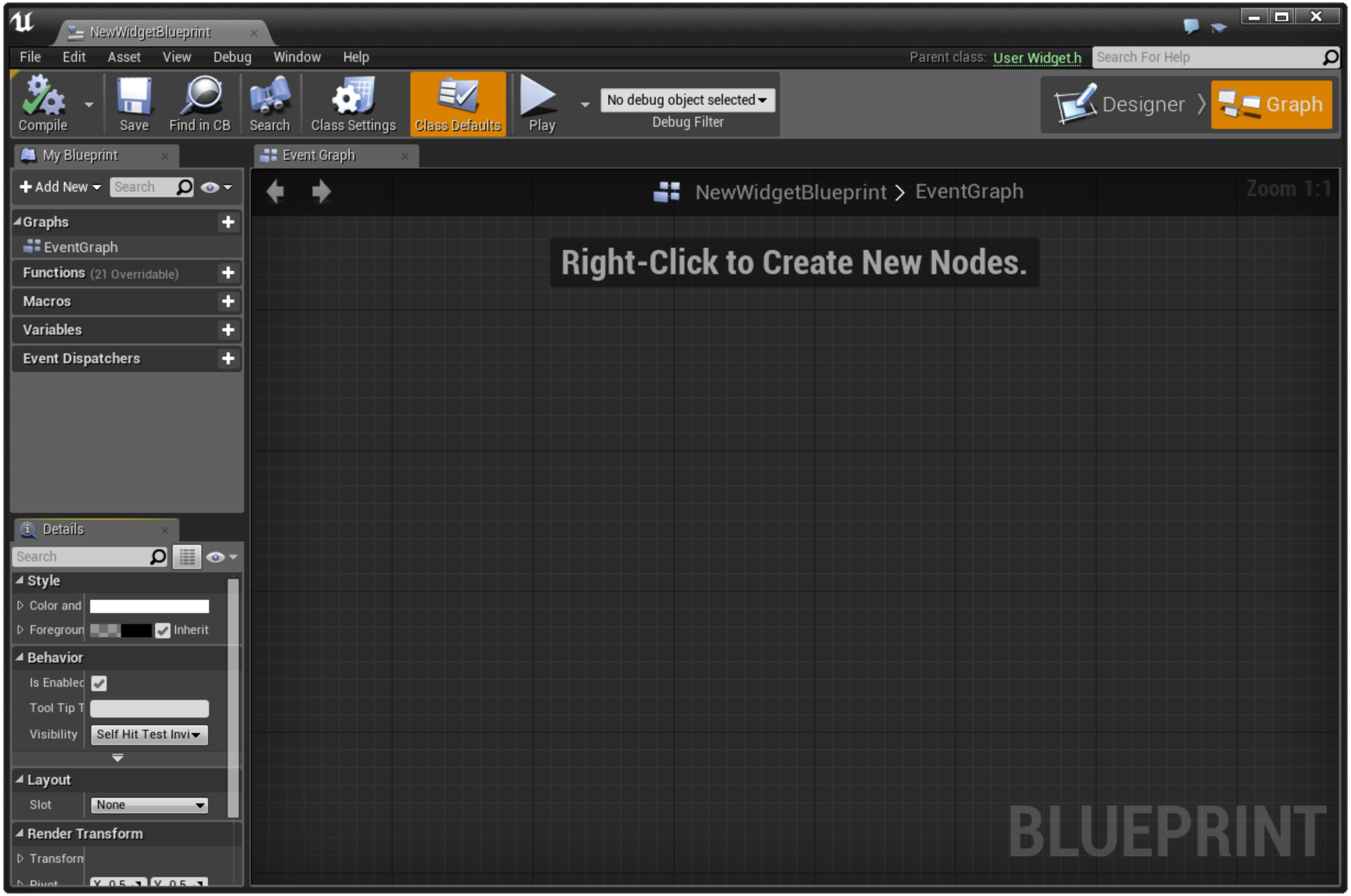
GUIS IN UNREAL

- Slate is UE5’s custom UI programming framework
    - Unreal editor is built in Slate
    - Written in C++
    - Can customize editor panels or be used in-game
    - Primarily used for tools-building
- UMG (Unreal Motion Graphics) is UE5’s visual UI authoring tool
    - Built using Widget Blueprints
    - Blueprint includes layout mode and event graph mode for reacting to inputs

WIDGET BLUEPRINTS

- Similar concept to Animation Blueprints
  - Specialized graph and visualization functionality built for user interface elements
- Built-in functionality for:
  - Constraints
  - Animations
  - Events
  - Scaling
  - Styling
  - etc…

WIDGET BLUEPRINT EDITOR



Parent class: User Widget.h

No debug object selected

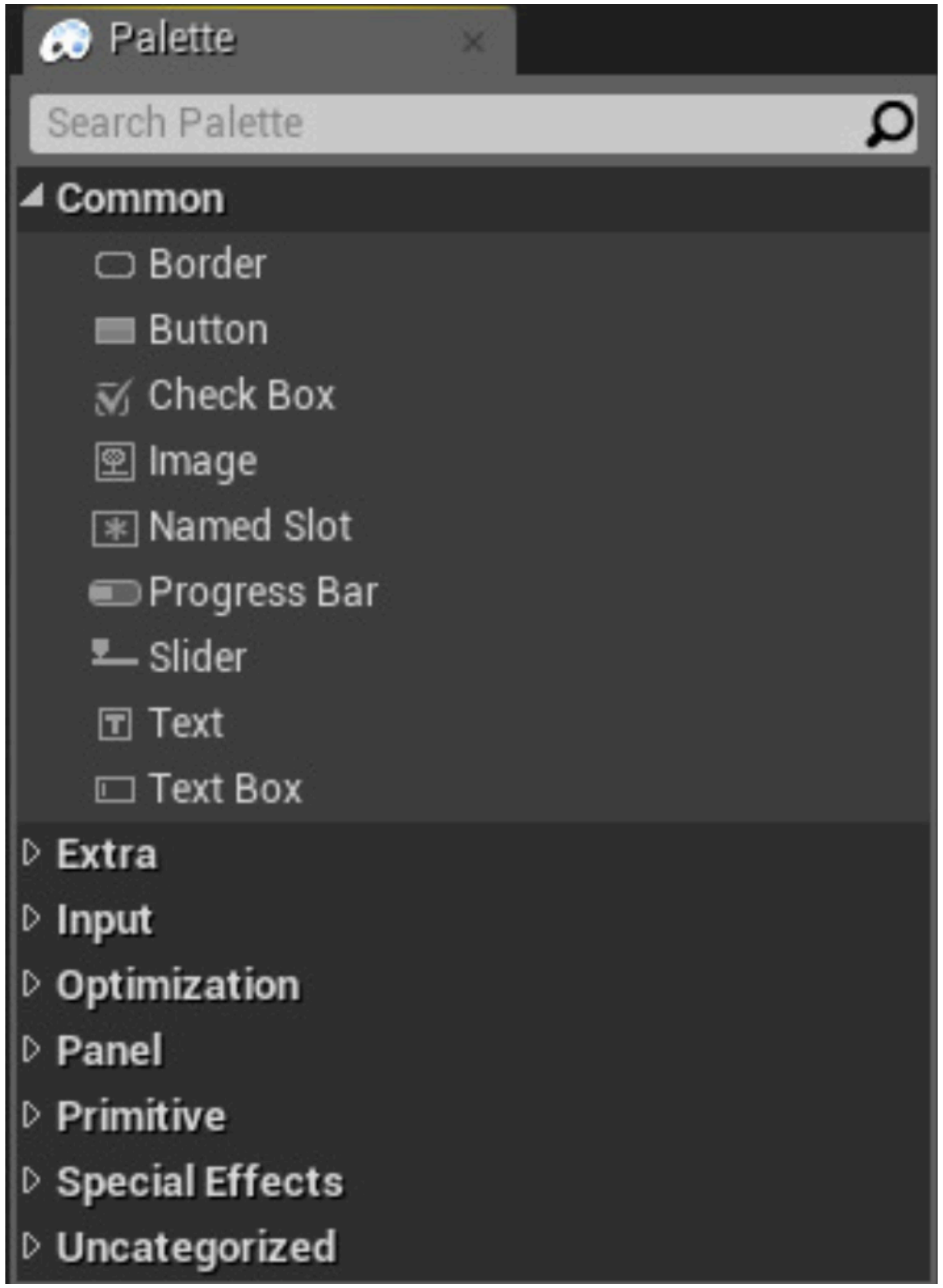
Right-Click to Create New Nodes.

Visibility: Self Hit Test Invi

Slot: None

WHAT ARE WIDGETS?

- Widgets are the common GUI elements used to convey information and provide events
- UMG widget examples:
  - Border
  - Button
  - Image
  - Checkbox
  - Text
  - Slider
  - etc…

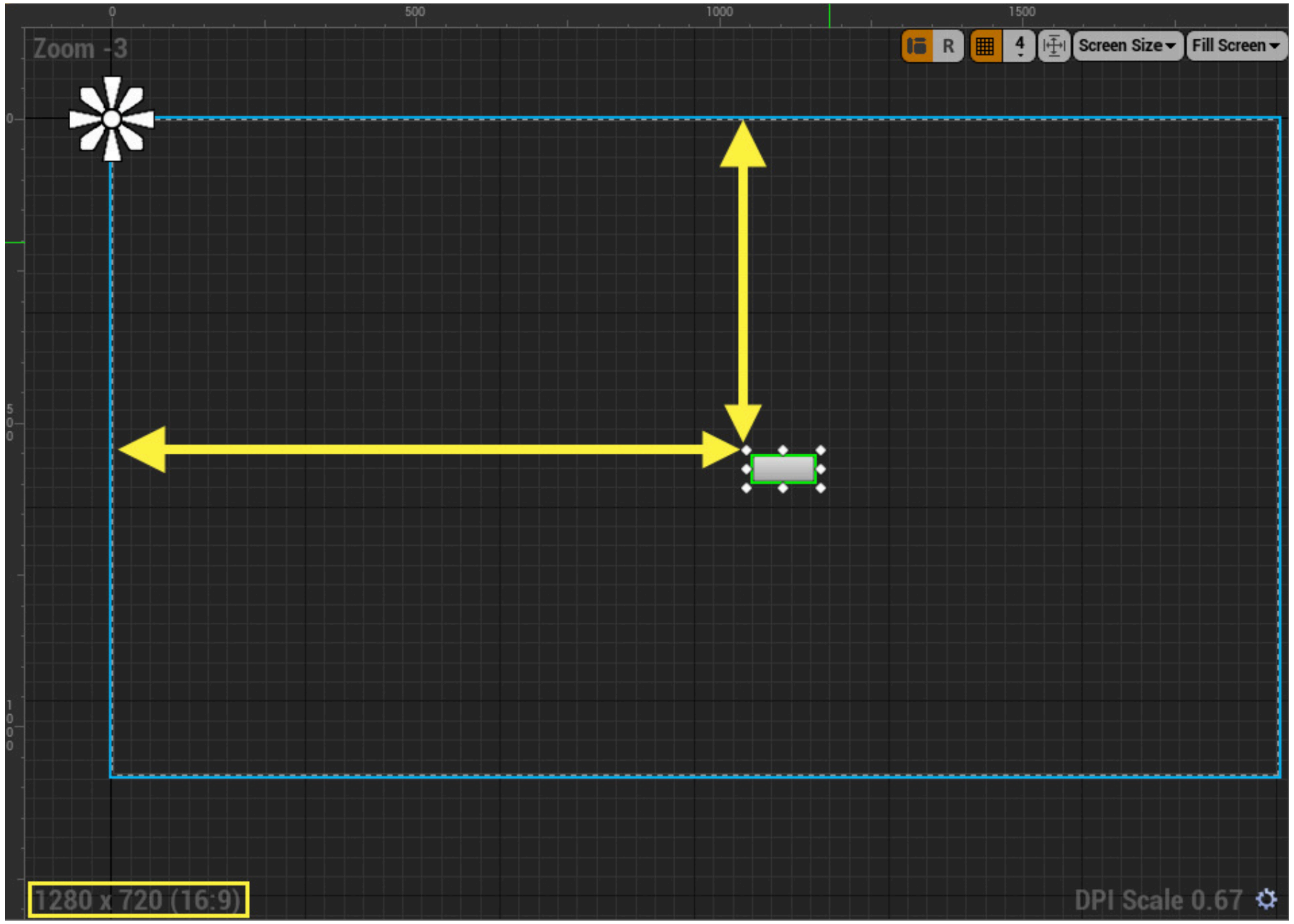


HOW CAN WE BE RESOLUTION INDEPENDENT?

- Resolve widget placement using **constraints**
- Layout can be treated as a system of linear equations and constraints
  - Treat as an optimization problem (minimize constraint violations)
  - Resolve using a linear objective function
- Soft constraints (i.e. requested constraints that can be violated if necessary to find a solution) can be violated in non-uniform ways
  - Quadratic objective functions better handle the minimization of error
- Constraint solving can decrease responsiveness
- Constraint solving allows for static analysis of violations

ANCHORS

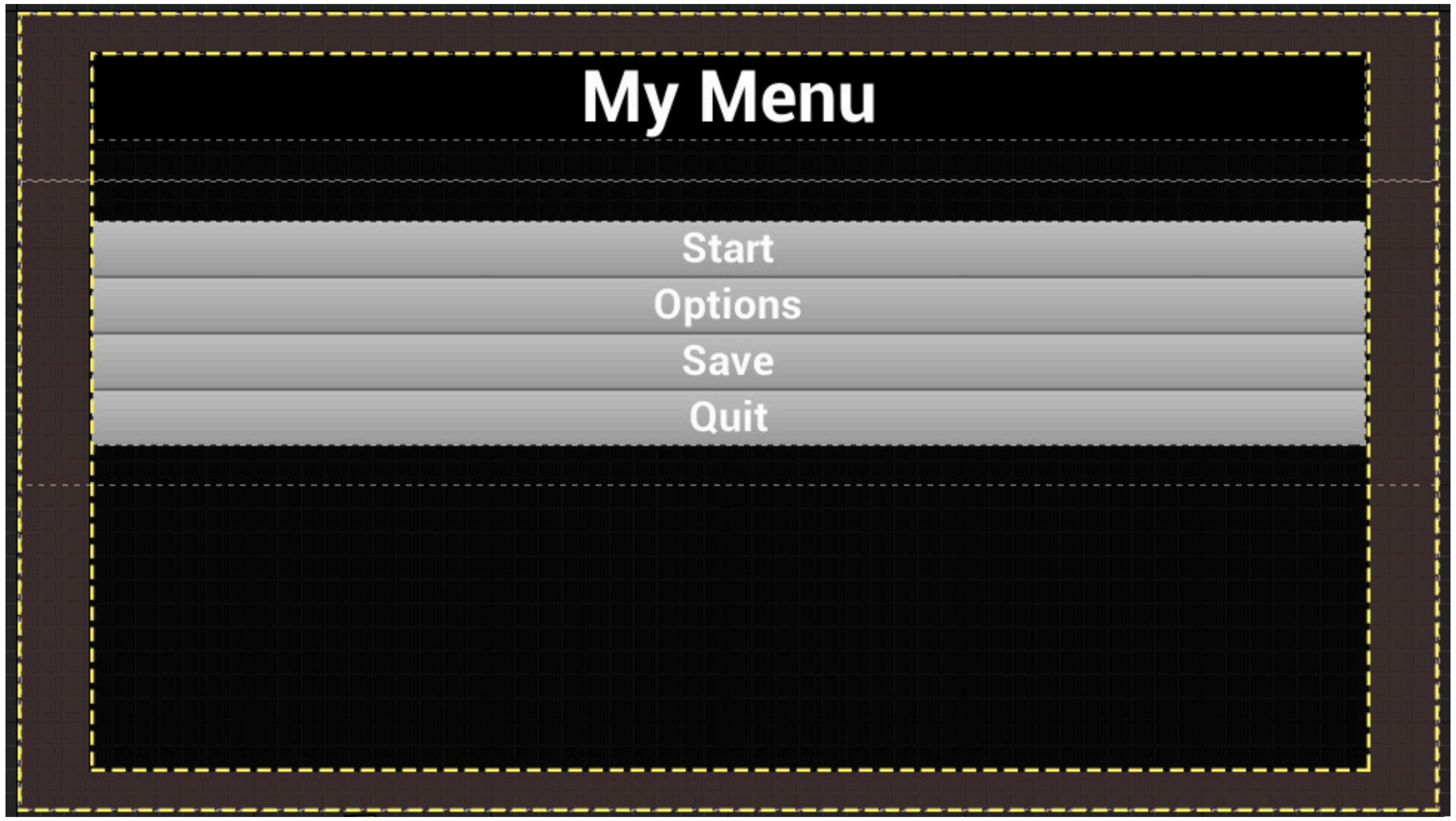
- Anchors define desired position within a Canvas Panel
  - Normalized between 0 and 1 for min and max
  - Origin (0, 0) is in upper left corner
- Can place anchor manually within the scene



Widget anchored to upper left corner

SAFE ZONES

- Specialized widgets that handle “unsafe” regions per device and resolution
  - e.g. edges of a TV, under the home bar of an iPhone, etc…
- Elements in a Safe Zone widget will adjust according to device resolution and orientation to ensure all screen elements are visible



Outer region is “unsafe” for given device preview

FONTS AND LOCALIZATION

- UE5 comes with several default fonts but they assume English language characters
- Possible to import custom fonts as assign them to text assets
- Actual text displayed should be saved in `FText` structs
  - Implemented with Shared Reference Pointers
  - Efficient checks for dirty in cache
  - Efficient serialization/network support
- `LOCTEXT` family of macros handles localization
  - Includes namespace, key, and source string

WHAT IS LOCALIZATION AND WHY DOES IT MATTER?

- Localization is the process of updating a game to be relevant to a region's audience
  - Respecting a country's censorship laws
  - Updating voice acting to be in the local language(s)
  - Updating text to be in the local language(s)
- Good localization ensures the cultural and language contexts are successfully conveyed

Japanese Name: Naruhodō Ryūichi

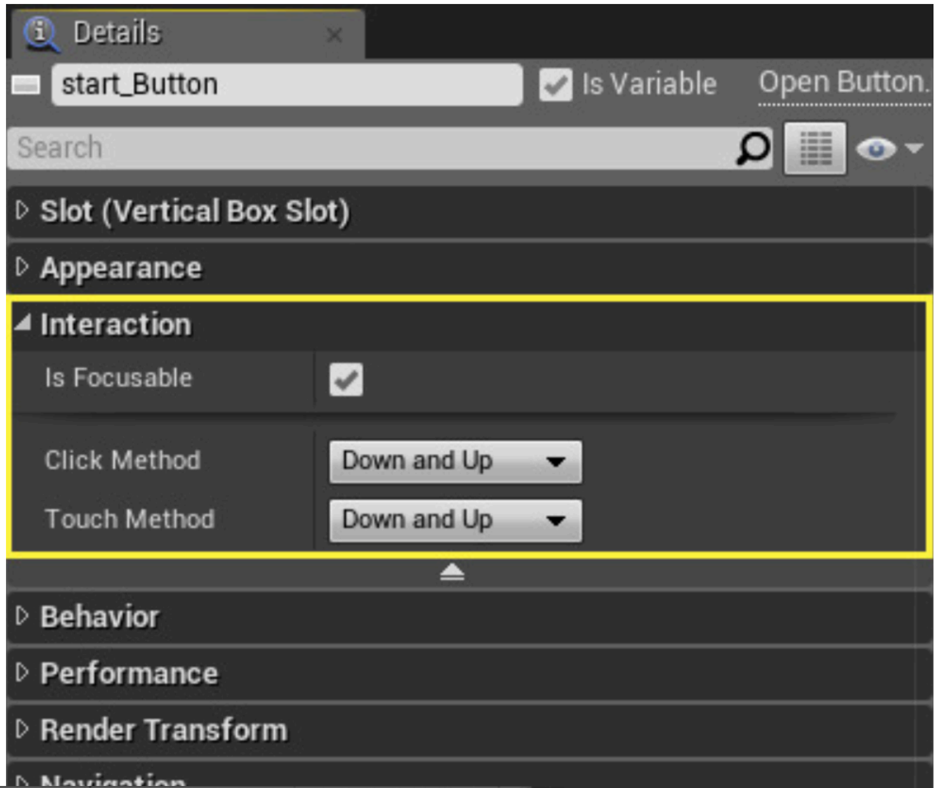
English Name: Phoenix Wright

ACCESSIBILITY

- UE5 supports screen readers with common widget elements
    - Allows 3rd party screen readers to access written data and “say” what is written
- Must enable screen reader support in project then specify which widgets should be accessible
- Can add support for custom widgets via C++
    - We’ll come back to the underlying C++ a bit later…

UMG EVENTS

- Similar flow to standard Blueprint Event Graphs
  - Focused on UI elements and interactions
- Bindable events use a single handler
- Multicast events connect widget ala BP

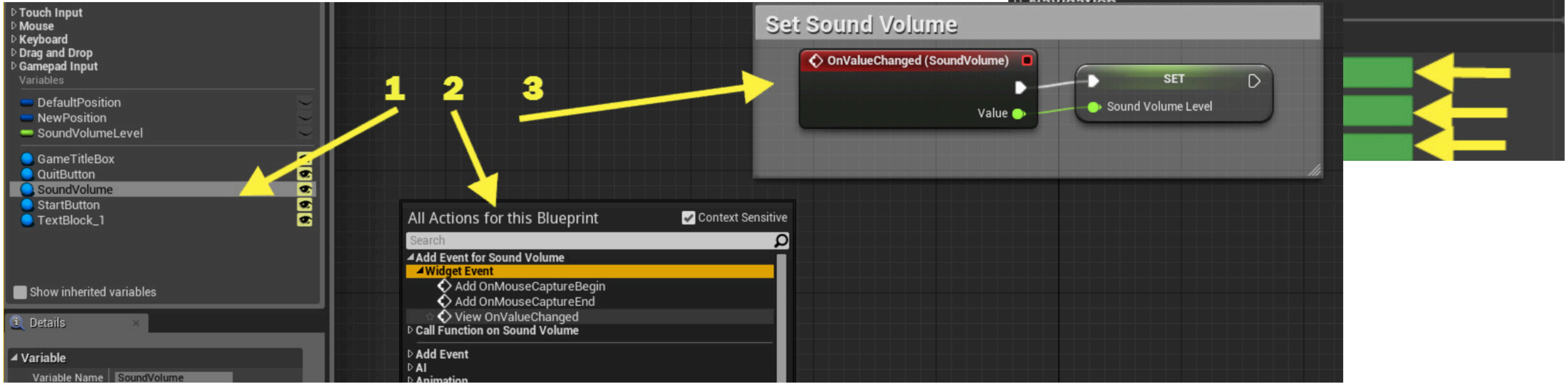


start_Button ☑ Is Variable Open Button.

Is Focusable ☑

Click Method: Down and Up

Touch Method: Down and Up



☐ Show inherited variables

Variable Name: SoundVolume

☑ Context Sensitive

WAIT…IS THIS ALL STUFF WE'RE SUPPOSED TO DO?

- Not really…UI artists and designers primarily work in these systems
  - Requires a lot of very specialized knowledge to be competent
- That said UI/UX programmers often need to assist artists and designers with their workflow
  - Take Blueprints created by artists/designers and translate them into efficient C++ implementations
  - Build underlying tools and systems to assist artists and designers

USING UMG WITH C++

- Ideally we want a C++ base with UMG Blueprint functionality built on top of it
  - More efficient to run
  - Cleaner to use
  - Less merge conflicts!
- Need to add UMG and Slate to our included modules (e.g. the libraries our project depends on)
- Need to add the necessary includes to the project header

USING GUI MODULES

- Under `ProjectName.Build.cs`:
  - Add `“UMG”` to `PublicDependencyModuleNames.AddRange()`
  - Add `“Slate”, “SlateCore”` to `PrivateDependencyModuleNames.AddRange()`
- In ProjectName.h add the following includes:
  - `#include "Runtime/UMG/Public/UMG.h"`
  - `#include "Runtime/UMG/Public/UMGStyle.h"`
  - `#include "Runtime/UMG/Public/Blueprint/UserWidget.h"`
  - `#include "Runtime/UMG/Public/Slate/SObjectWidget.h"`
  - `#include "Runtime/UMG/Public/IUMGModule.h"`

CREATING WIDGET CLASSES

- Inherit from `UserWidget` to allow extensions to Blueprint
  - Create functions, properties, and events in either C++ or BP as we've seen previously
- Connect widgets to PlayerControllers to have them display **for that player**
  - `MyWidget->AddToViewport();`
- Can create a widget using `CreateWidget<MyWidget>(this, MyWidgetBP);`
- Can define `MyWidgetBP` via Blueprint or using `FClassFinder` in the constructor

USING FCLASSFINDER

- In .h

```
    UPROPERTY(...)
    TSubclassOf<MyWidget> MyWidgetBP;
```

- In .cpp

```
    static ConstructorHelpers::FClassFinder<MyWidget>
BlueprintClass(TEXT(“/Path/to/Blueprint/Reference“));
    if (BlueprintClass.Succeeded())
        MyWidgetBP = BlueprintClass.Class;
```

FCLASSFINDER VS FOBJECTFINDER

- Provide functionality for finding either a `UClass` or a `UObject` respectively
- `UClass` derives from `UObject`, so `FObjectFinder` is more general
- Note: “`/Path/to/Blueprint/Reference`“ refers to the blueprint asset whereas “`/Path/to/Blueprint/Reference_C`” refers to the class object
- In many cases, both finders are valid ways of finding either the object itself or the class object

CONNECTING WIDGETS TO C++

- Create a UPROPERTY with specifier `meta = (BindWidget)`
    - Name of widget in .h **must match** name in UMG!
- Add delegate function pointers in `Initialize()`
    - `MyButton->OnClicked.AddDynamic(this, &MyClass::OnClickedFunction);`
- Can create C++ functionality for all Widgets (including sub-widgets of other widgets)
    - Widget composition can get quite complex, so take time to reason through the UX functionality before building

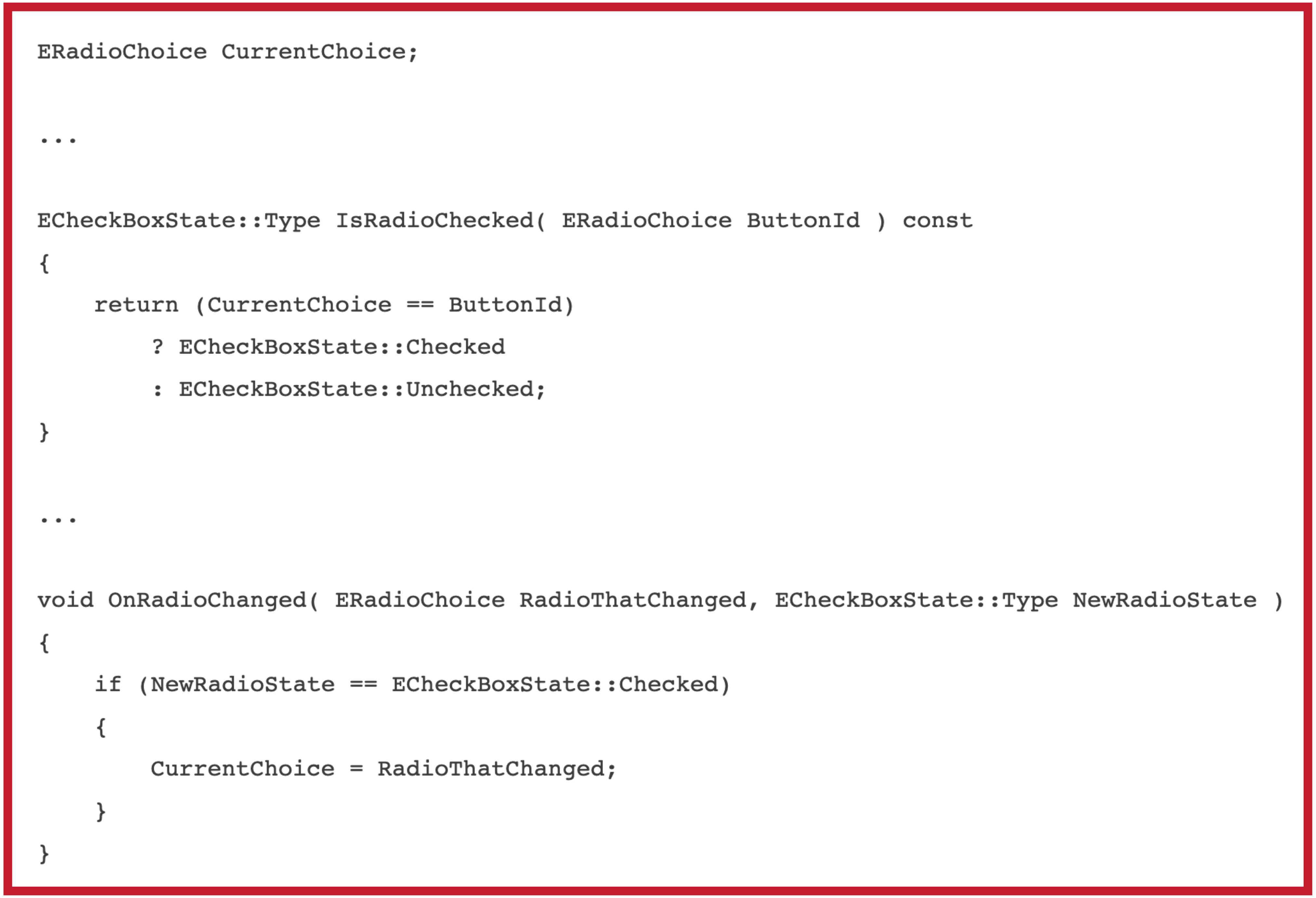
WIDGET COMPONENTS

- 3D Widgets that can be placed into a world by attaching them to actors
  - Same idea as any other component
  - Derive from UMeshComponent -> UPrimitiveComponent -> USceneComponent -> UActorComponent
- Must include necessary modules in `Build.cs` to create them in C++
- Useful for diegetic content (e.g. UI that exists in the world) and context-sensitive content (e.g. UI that exists for the player but only in certain states)
- Many built-in functions for determining how to display and where (i.e. across a network)

SLATE

- Custom UI framework for Unreal
  - Built as a declarative UI-description language in C++
- Used to build Unreal's Editor!
  - Ideal choice for building UE5 editor plugins
- Can be used to build in-game widgets to avoid dealing with UMG (which is notably built on Slate)
  - UMG is a WYSIWYG; Slate resembles a mark-up language
  - Not particularly recommended though…

SLATE EXAMPLES



```
ERadioChoice CurrentChoice;

...

ECheckBoxState::Type IsRadioChecked( ERadioChoice ButtonId ) const
{
    return (CurrentChoice == ButtonId)
        ? ECheckBoxState::Checked
        : ECheckBoxState::Unchecked;
}

...

void OnRadioChanged( ERadioChoice RadioThatChanged, ECheckBoxState::Type NewRadioState )
{
    if (NewRadioState == ECheckBoxState::Checked)
    {
        CurrentChoice = RadioThatChanged;
    }
}
```

Define radio buttons as an enum of checkboxes

SLATE EXAMPLES

```
FMenuBarBuilder MenuBarBuilder( CommandList );
{
    MenuBarBuilder.AddPullDownMenu( TEXT("Menu 1"), TEXT("Opens Menu 1"), FNewMenuDelegate::CreateRaw( &FMenus::FillMenu1Entries ) );

    MenuBarBuilder.AddPullDownMenu( TEXT("Menu 2"), TEXT("Opens Menu 2"), FNewMenuDelegate::CreateRaw( &FMenus::FillMenu2Entries ) );
}

return MenuBarBuilder.MakeWidget();
```

A menu example

SLATE ARCHITECTURE DESIGN

- Goals are to:
  - Have easy access to data and models
  - Allow procedural UI generation
  - Support for animation and styling
  - Limit ability to mess up UI descriptions
- Slate is compile-time checked
- Two passes: caching desired widget size, and arranging children accordingly

SLATE ARCHITECTURE CHOICES

- Avoid opaque caches and duplicated state over CPU concerns
- All current layout based on programmer settings rather than previous layout state
- Prefer polling data whenever possible
- If necessary, use of delegates to retrieve and modify data from the model if state is not drastically changing
- If necessary, use of delegates with low-grain invalidation to modify data if state has drastically changed
    - e.g. in Blueprints, changes to the Event Graph results in all widgets being cleared and recreated

ASSUMPTIONS (FOR GOOD OR ILL)

- Developer side performance:
  - Programmers are expensive; CPUs are fast and cheap
- Gameplay side performance:
  - UI complexity is bound by number of live widgets, so avoiding live widgets off-screen limits performance dips
  - If players have big screens, they also have beefy machines to drive those screens

REFERENCES

- UMG Documentation <https://docs.unrealengine.com/en-US/Engine/UMG/index.html>
- Using Unreal Motion Graphics (UMG) with C++ <https://www.orfeasel.com/using-unreal-motion-graphics-umg-with-c/>
- UWidgetComponent Documentation<https://docs.unrealengine.com/en-US/API/Runtime/UMG/Components/UWidgetComponent/index.html>
- Slate Documentation <https://docs.unrealengine.com/en-US/Programming/Slate/index.html>