Programowanie Gier Komputerowych

Wykład 9: User Interface

User Interface

User Interface to wszystko, co gracz widzi na ekranie jako warstwę informacyjną:

- HUD (zdrowie, mana, amunicja),
- Menu główne / pauzy,
- Ekrany ekwipunku, questów,
- Komunikaty (kombo, misje, tutoriale).

W Unreal Engine za UI odpowiada głównie system **UMG – Unreal Motion Graphics**.

Architektura UI w Unreal Engine

Unreal Engine oferuje trzy warstwy UI:

- **UMG (Unreal Motion Graphics):** system graficzny UI (Blueprint + Edytor).
- **Slate (C++):** niskopoziomowy framework UI (mniej wygodny, bardziej wydajny).
- **Widget Blueprinty:** kluczowy komponent – tworzy się w edytorze, z logiką eventów.

W zaawansowanych projektach:

- używa się **kompozycji widgetów**,
- **delegatów i eventów** do komunikacji,
- **controllerów UI** do zarządzania widokami.

Co to jest UMG?

UMG to system do tworzenia UI za pomocą:

- **Widgetów** – komponentów UI (np. tekst, obraz, przycisk),
- **Blueprintów widgetów** (UserWidget),
- **Hierarchy + Layout system** – organizacja interfejsu.

Widgety można dynamicznie tworzyć, usuwać, ukrywać, podmieniać — UI to pełnoprawna część gry.

Czym jest Slate?

Slate to **C++-owy framework UI**, który działa pod spodem systemu UMG (Unreal Motion Graphics). UMG to graficzna nakładka na Slate, a wszystkie widgety w UMG są tak naprawdę Slate-widgetami z opakowaniem.

Cechy Slate:

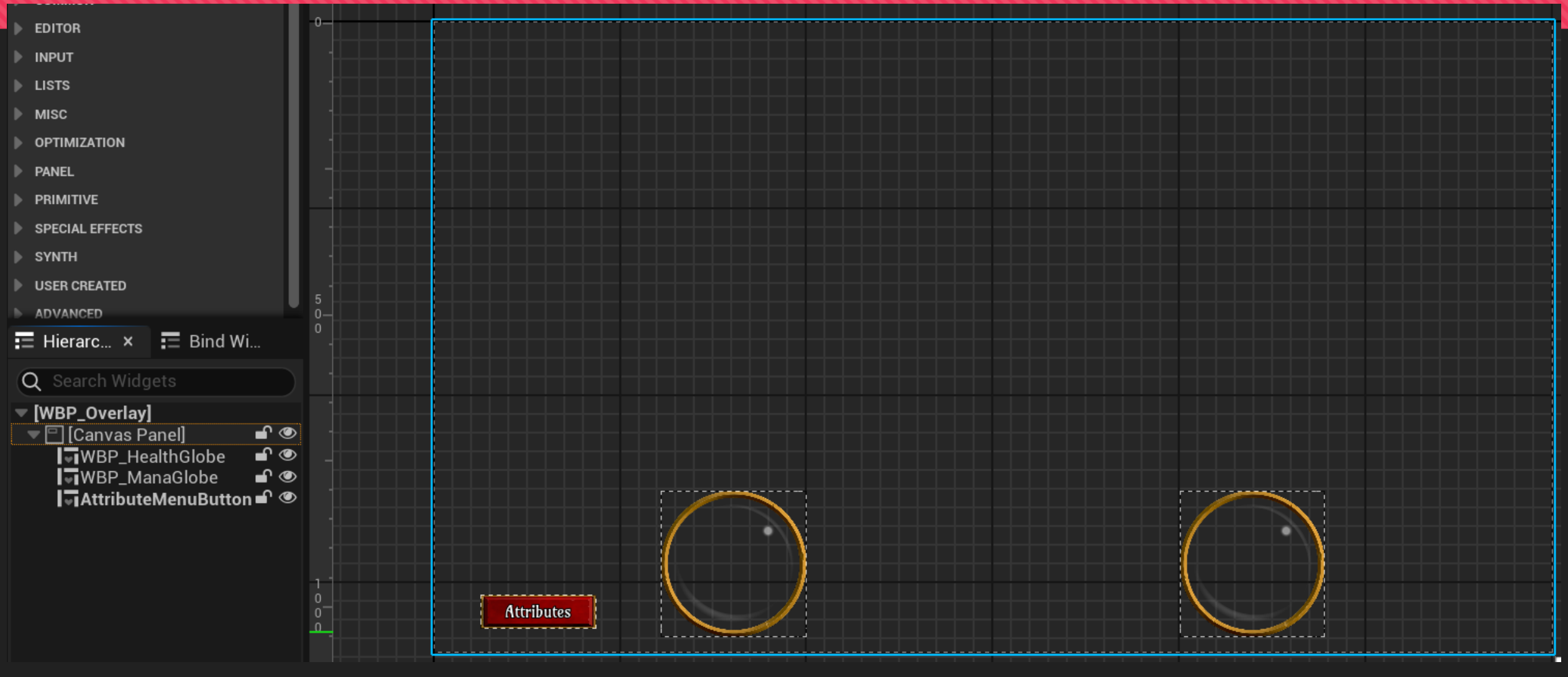
- Napisany w **czystym C++**, bez Blueprintów.
- **Bardzo wydajny** (brak kosztów edytora, lepsza kontrola nad rysowaniem).
- **Elastyczny i potężny** – możesz tworzyć własne typy widgetów od zera.
- Używany w **interfejsie edytora Unreal Engine**.

Komponenty UI – co wybrać i kiedy?

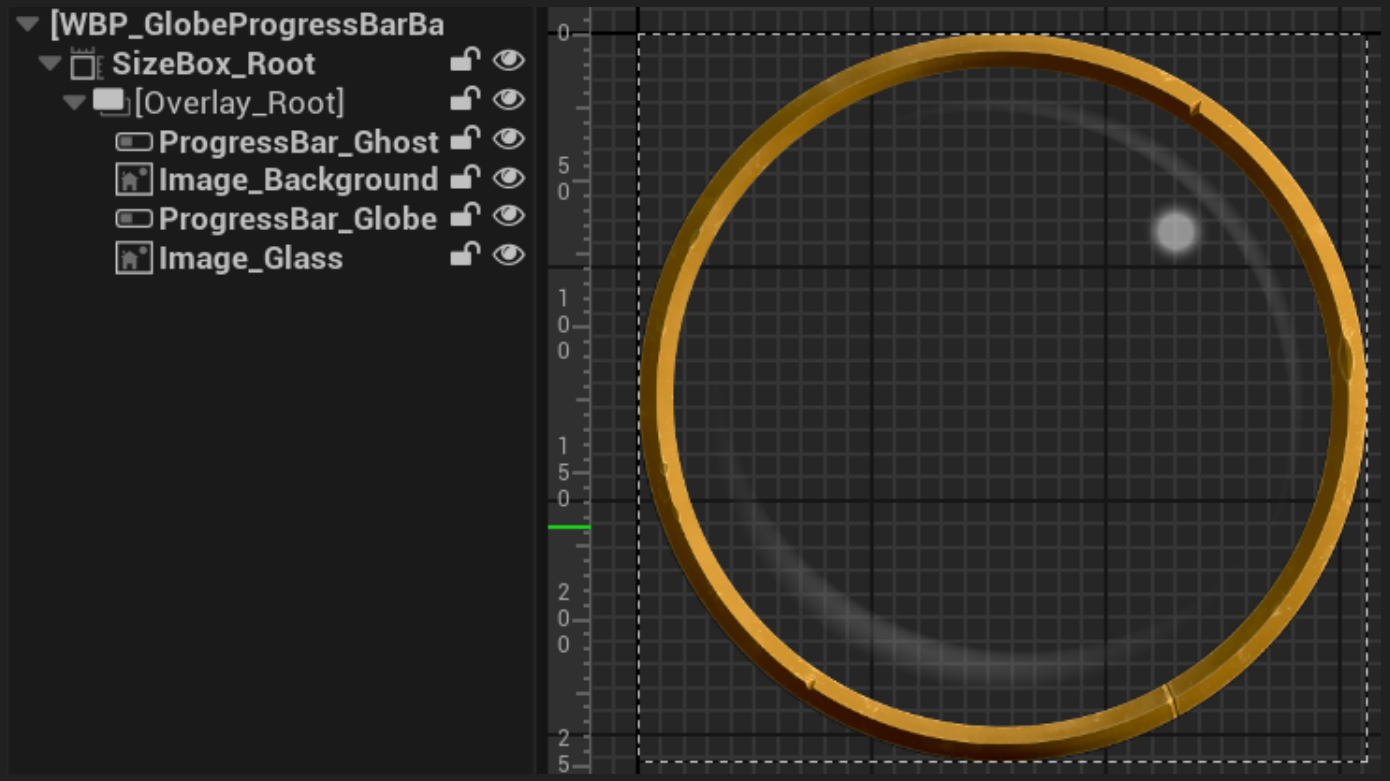
Komponent	Zastosowanie
Canvas Panel	Swobodne rozmieszczanie elementów (UI o stałej pozycji).
Size Box	Wymuszanie rozmiaru niezależnie od zawartości.
Horizontal/Vertical Box	Układanie wielu elementów w rzędzie/kolumnie.
Overlay	Nałożenie wielu warstw (np. tekst + tło + ikona).
ScrollBox	Długie listy (np. ekwipunek, logi).
WidgetSwitcher	Przełączanie widoków (np. różne zakładki).

Dobre praktyki:

- Unikaj **Canvas Panel** jako głównego layoutu w dynamicznych UI.
- Preferuj kontenery do responsywności.


EDITOR
INPUT
LISTS
MISC
OPTIMIZATION
PANEL
PRIMITIVE
SPECIAL EFFECTS
SYNTH
USER CREATED
ADVANCED
Hierarc...
Bind Wi...
Search Widgets
[WBP_Overlay]
[Canvas Panel]
WBP_HealthGlobe
WBP_ManaGlobe
AttributeMenuButton
Attributes

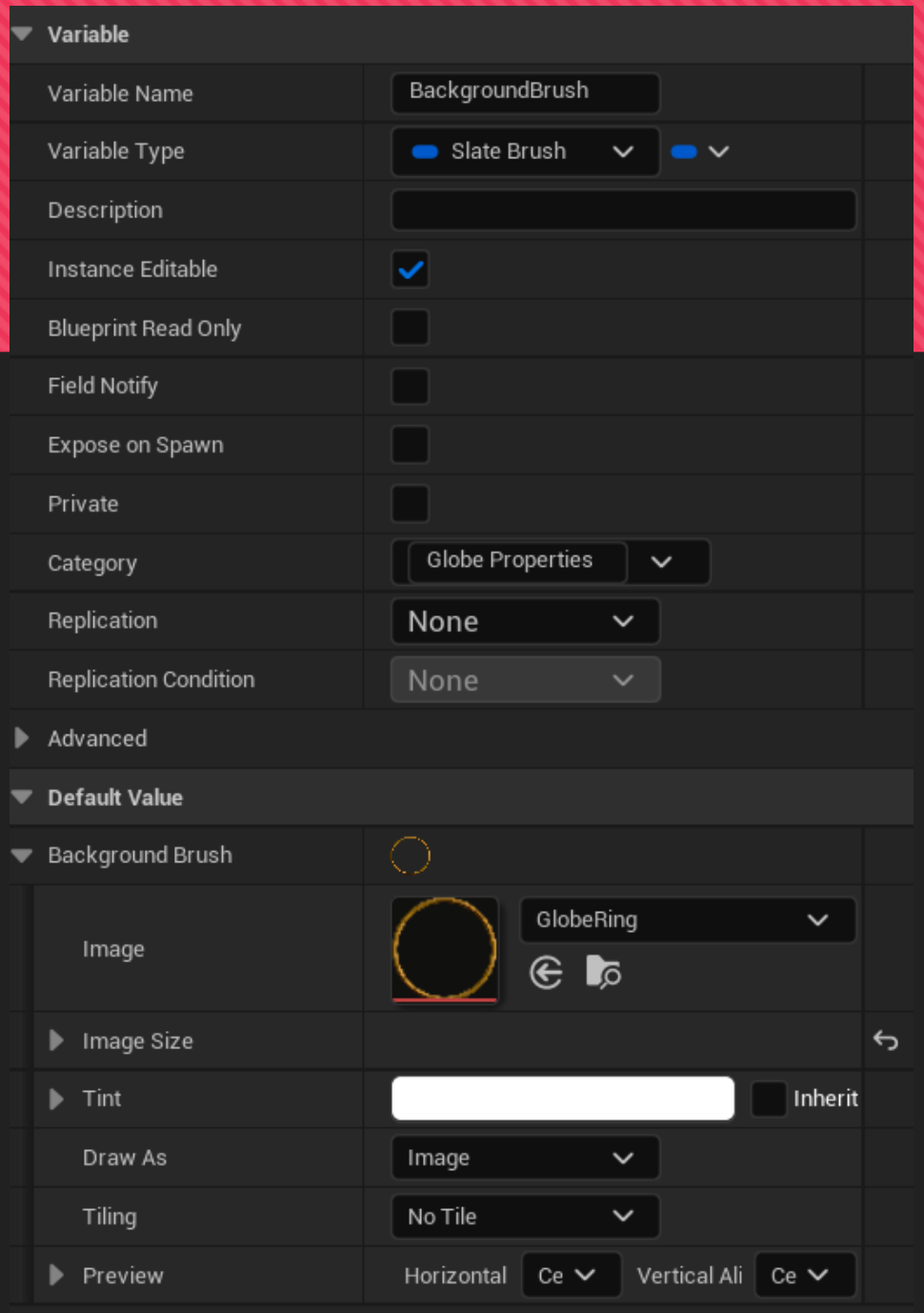

[WBP_EffectMessage]
[Overlay]
[HorizontalBox_Root]
Image_Icon
[Spacer]
[Text_Message] "Pi
Picked up a Health Potion

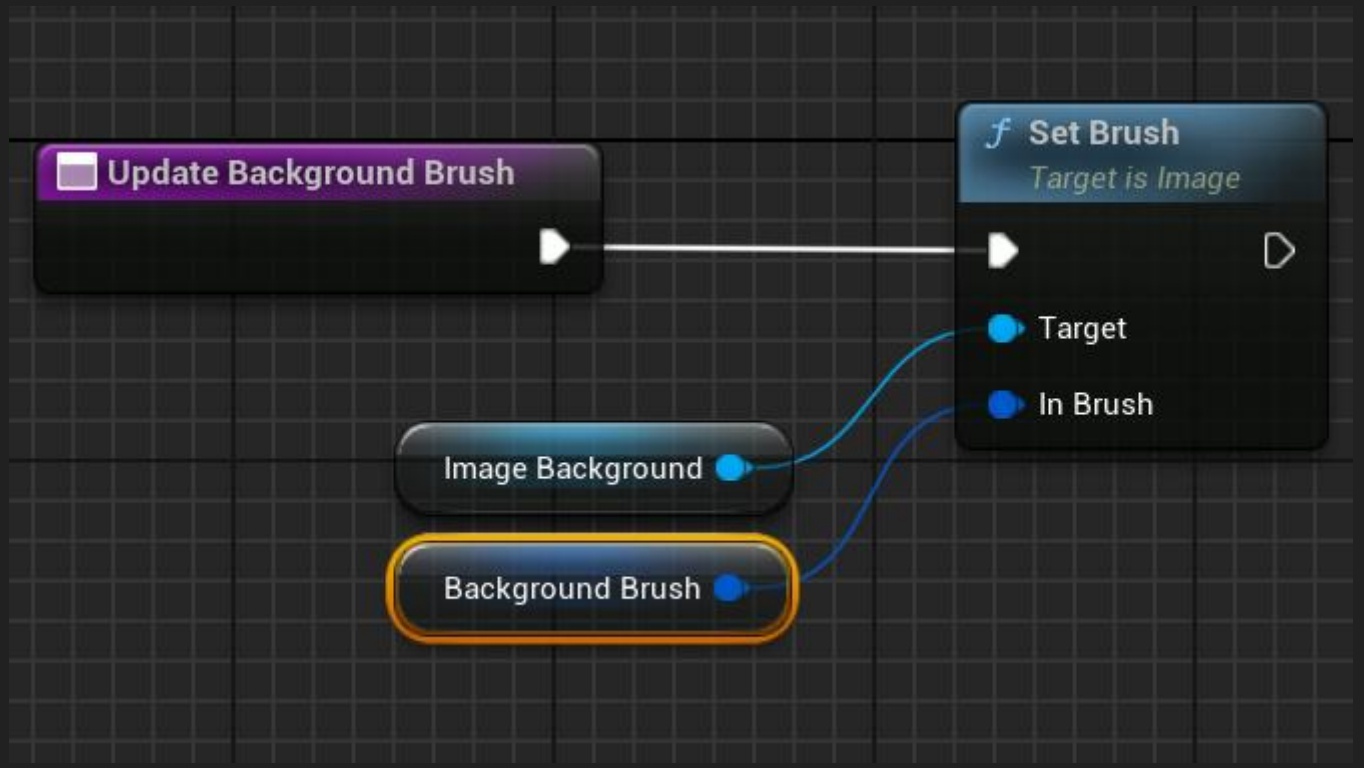

[WBP_GlobeProgressBarBa
SizeBox_Root
[Overlay_Root]
ProgressBar_Ghost
Image_Background
ProgressBar_Globe
Image_Glass

Stylowanie UI – hierarchie wyglądu

- **Widget Style Assets** – styl tekstu, przycisków, suwaków.
- **Slate Brush** – definiuje grafikę tła/przycisku (np. z assetów).
- **Font Family + Size** – czcionki jako assety (TTF/OFT).
- **Globalne style** – jako **DataTable** lub **StyleSet**.

Dobrze zorganizowany UI ma **oddzielne warstwy stylu** – dzięki temu łatwo dostosować wygląd globalnie.


Variable
Variable Name
BackgroundBrush
Variable Type
Slate Brush
Description
Instance Editable
Blueprint Read Only
Field Notify
Expose on Spawn
Private
Category
Globe Properties
Replication
None
Replication Condition
None
Advanced
Default Value
Background Brush
Image
GlobeRing
Image Size
Tint
Inherit
Draw As
Image
Tiling
No Tile
Preview
Horizontal
Ce
Vertical Ali
Ce


Update Background Brush
Set Brush
Target is Image
Target
In Brush
Image Background
Background Brush

Systemy modularne – tworzenie własnych widgetów

Przykład: **UInventorySlotWidget**, **UAbilityIconWidget**

- Mają własną logikę (np. OnHover, OnClicked)
- Mogą emitować delegaty (OnSlotSelected).
- Łatwe do użycia w listach, drag & drop itd.

Tworzenie własnych klas widgetów pozwala utrzymać porządek i rozszerzać UI bez duplikacji.

Appearance

Style

Normal

Hovered

Pressed

Disabled

Events

On Clicked

On Pressed

On Released

On Hovered

On Unhovered

Komunikacja UI ↔ gra (Blueprint + C++)

W typowym przepływie danych UI komunikuje się z:

- PlayerController (np. do pokazania menu).
- Character (np. do użycia umiejętności).
- GameState, GameInstance (dane globalne).

Komunikacja UI ↔ gra (Blueprint + C++)

Mechanizmy:

- Bind – np. Bind Text to Function (rzadziej polecane przy dużych UI - kosztowne)
- Event Dispachers (Delegaty) – polecane do przekazywania akcji z widgetów.
- Interface'y – dobre do komunikacji między UI a gameplayem.

Animacje UI – kluczowe elementy

Animacje UI są:

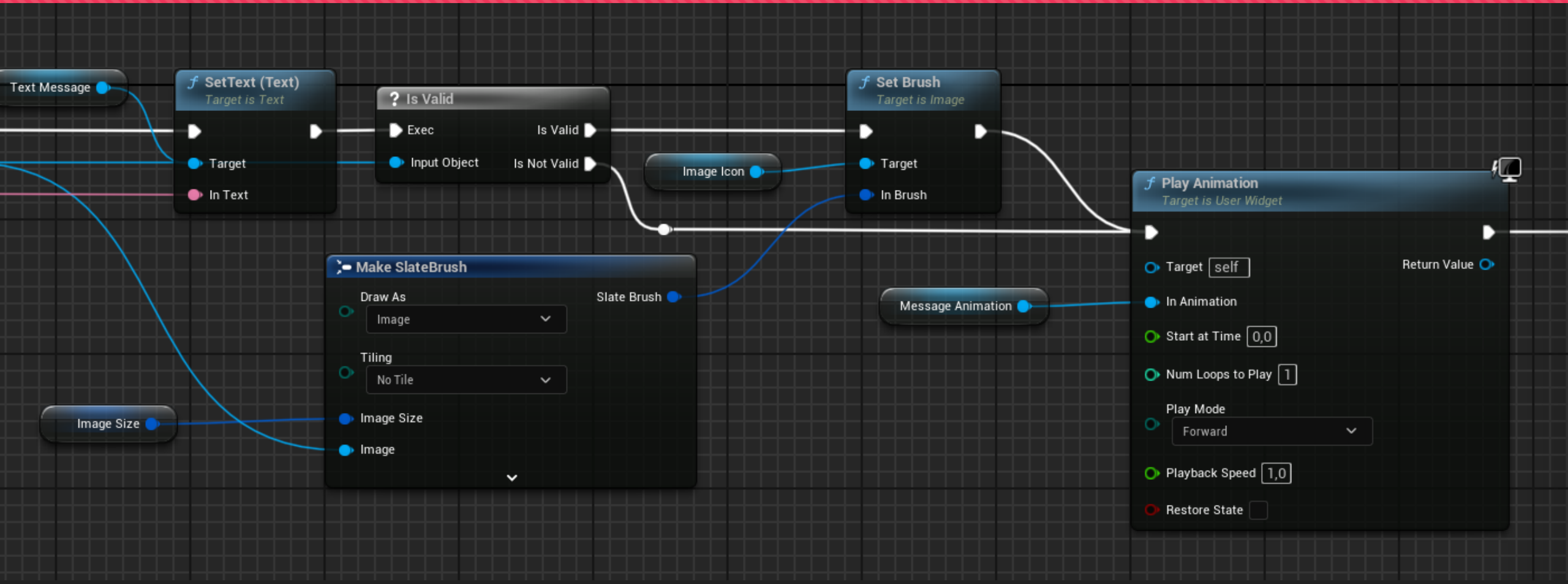
- Wizualną informacją zwrotną (hover, klik, otrzymanie obrażeń).
- Elementem stylu (wjeżdżające panele, przyciski z "sprężyną").

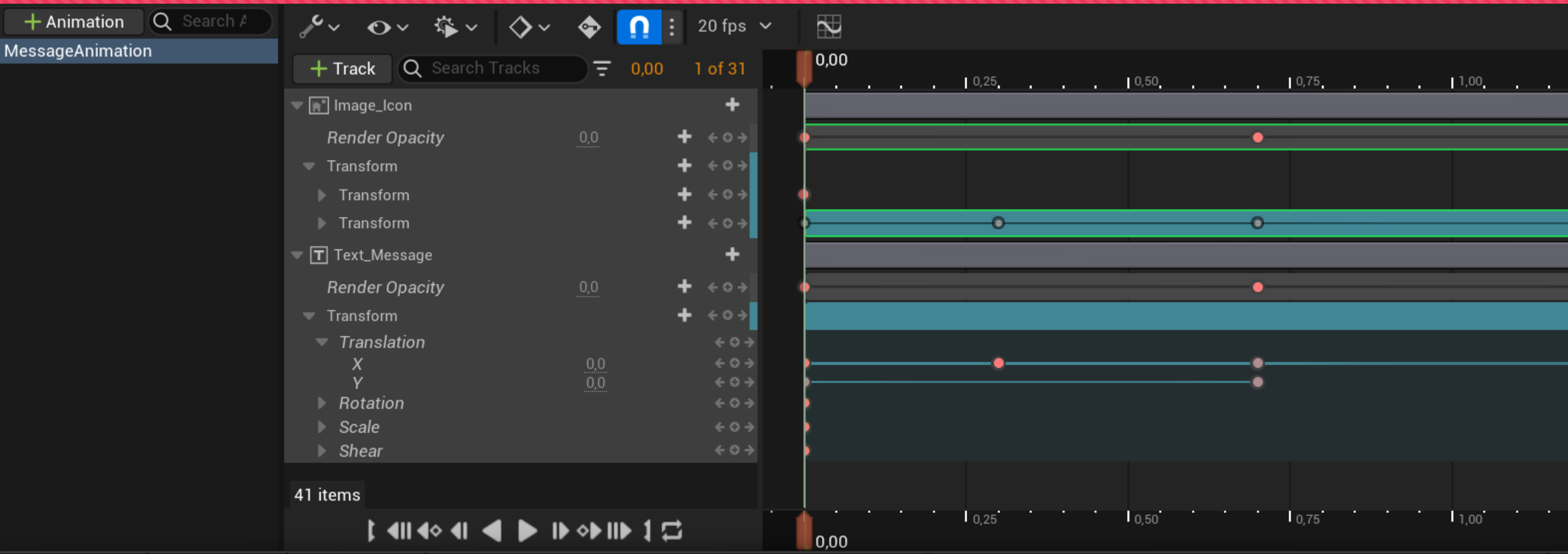
W Unreal:

- Używamy **Timeline w UMG**.
- Sterowanie z Blueprinta: Play Animation, Play Reverse, Bind on Finished.

Zasada: animacje powinny być szybkie (150–400ms), czytelne, nienachalne.

Animacje UI


Text Message
SetText (Text)
Target is Text
Target
In Text
Is Valid
Exec
Input Object
Is Valid
Is Not Valid
Image Icon
Set Brush
Target is Image
Target
In Brush
Play Animation
Target is User Widget
Target
self
Return Value
In Animation
Message Animation
Start at Time
0,0
Num Loops to Play
1
Play Mode
Forward
Playback Speed
1,0
Restore State
Make SlateBrush
Draw As
Image
Slate Brush
Tiling
No Tile
Image Size
Image
Image Size


Animation
Search A
MessageAnimation
20 fps
Track
Search Tracks
0,00
1 of 31
Image_Icon
Render Opacity
0,0
Transform
Transform
Transform
Text_Message
Render Opacity
0,0
Transform
Translation
X
0,0
Y
0,0
Rotation
Scale
Shear
41 items
0,00
0,25
0,50
0,75
1,00
0,00
0,25
0,50
0,75
1,00
Content Drawer
Animations
Output Log
Cmd
Enter Console Command

UI i wejście gracza (Input Routing)

W kontekście UI, Unreal ma 3 tryby inputu:

- **UI only** – tylko mysz / touch działa (menu).
- **Game only** – gameplay, UI nieaktywny.
- **Game and UI** – oba naraz (HUD + celowanie).

Używamy **SetInputMode…()** + **ShowMouseCursor**

Zmieniamy tryby w zależności od stanu gry (np. pauza => UI only)

WidgetSwitcher i zarządzanie widokami

Idealne do tworzenia interfejsów z zakładkami lub menu:

- Widget Switcher pozwala pokazywać tylko jedno z dzieci.
- Świetne do Tab UI, Ustawień, Menu głównego.

W dużych UI warto:

- Trzymać wszystkie podmenu w WidgetSwicher.
- Zarządzać nimi z poziomu UIManager (klasa pomocnicza do centralnego zarządzania UI)

Responsywność i skalowanie UI

UI musi działać w różnych rozdzielczościach i proporcjach ekranu dlatego też warto wykorzystać komponenty pomocnicze:

- Scale Box – automatycznie przeskaluje zawartość.
- Safe Zone – Marginesy pod różne platformy.
- Size Box, Uniform Grid – Sztywne lub elastyczne rozmieszczenie.

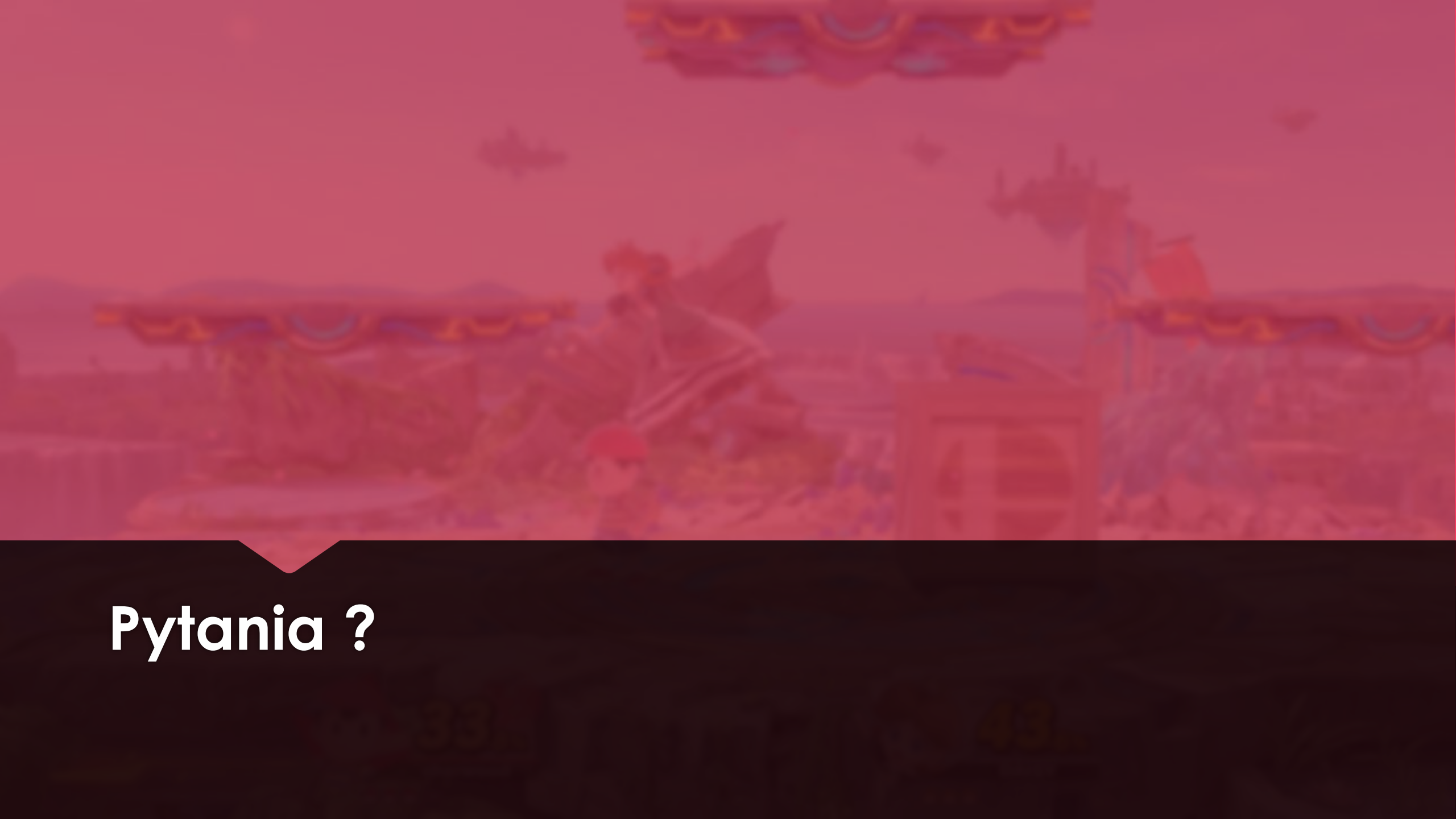
UI w grze 3D – Widget Components

Unreal Engine pozwala wyświetlać UI w przestrzeni gry:

- Widget Component – aktor renderujący UI jako mesh (np. panel nad NPC, Terminal, mapa taktyczna)

Używane do:

- Interaktywnych obiektów np. terminale
- HUDów 3D
- Znaczników nad głową gracza/NPC



Pytania ?

Dziękuję za uwagę!