Instrukcja laboratoryjna 6

Unreal Engine i środowiska immersyjne

1. Cel i kontekst ćwiczenia

Przez pięć poprzednich laboratoriów pracowaliśmy wyłącznie w środowisku Unity. To świadoma decyzja dydaktyczna: głębsze poznanie jednego narzędzia jest cenniejsze niż powierzchowna znajomość kilku. Jednak rynek przemysłowy jest pluralistyczny: różne projekty wymagają różnych narzędzi, a umiejętność adaptacji do nowego środowiska to kompetencja sama w sobie.

Unreal Engine (UE) firmy Epic Games to drugi co do popularności silnik czasu rzeczywistego na świecie. Jego pozycja jest szczególnie silna w branży gier AAA, produkcji filmowej i architektonicznej wizualizacji. Celem tego laboratorium nie jest pełne opanowanie UE, gdyż to wymagałoby osobnego kursu: lecz **wykształcenie zdolności transferu wiedzy**: przełożenie koncepcji znanych z Unity na nowe środowisko oraz rozpoznanie kluczowych różnic projektowych między obydwoma silnikami.

Po zakończeniu tego laboratorium student:

1. sprawnie porusza się po interfejsie edytora Unreal Engine i rozpoznaje odpowiedniki panelów znanych z Unity,
2. rozumie fundamentalne różnice architektoniczne między Unity a Unreal Engine (jednostki, pipeline renderowania, system assetów),
3. potrafi zaimportować modele 3D do projektu UE i skonfigurować podstawową scenę z oświetleniem i kamerą,
4. potrafi stworzyć prostą logikę aplikacji z użyciem systemu Blueprints bez pisania kodu C++,
5. potrafi krytycznie porównać oba silniki i argumentować wybór narzędzia do konkretnego rodzaju projektu,
6. rozumie zastosowania Unreal Engine poza branżą gier: w architekturze, filmie, inżynierii i szkoleniach.

2. Wprowadzenie teoretyczne

2.1. Historia i pozycja rynkowa Unreal Engine

Unreal Engine powstał w 1998 roku jako silnik gry **Unreal** studia Epic Games. [2] Od tamtej pory przeszedł siedem głównych rewizji. Unreal Engine 5, wydany produkcyjnie w 2022 roku, wprowadził przełomowe technologie renderowania: **Nanite** (wirtualizacja geometrii: renderowanie miliardów polygonów w czasie rzeczywistym) i **Lumen** (w pełni dynamiczne globalne oświetlenie bez konieczności pieczenia światła).

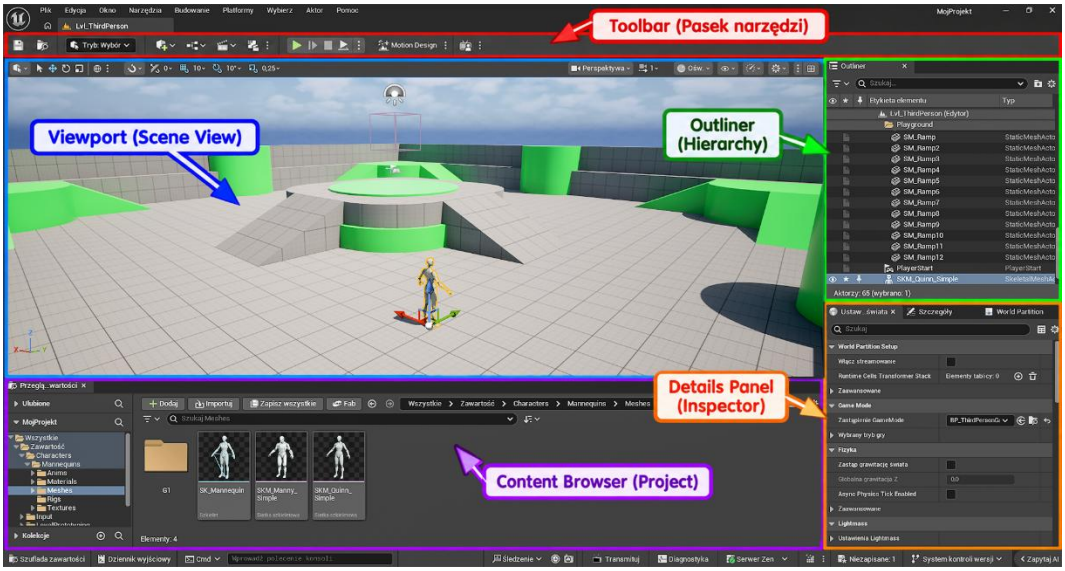
Wersja UE	Rok wydania	Przełomowe wprowadzenie
Unreal Engine 1	1998	Silnik gry FPS Unreal, zaawansowany jak na epokę renderer
Unreal Engine 3	2006	Unreal Script, materiały node-based, wsparcie konsol 7. generacji
Unreal Engine 4	2015	Pełny kod źródłowy na GitHub, Blueprints, PBR rendering
Unreal Engine 5	2022	Nanite, Lumen, World Partition, MetaHuman, PCG

Unreal Engine jest używany nie tylko w grach: **The Mandalorian** (Disney+) kręcony był na wirtualnym planie opartym na LED i UE, a takie firmy jak BMW, Volkswagen, Nokia i Audi stosują go do wizualizacji projektowych i symulacji szkoleniowych.

2.2. Architektura edytora Unreal Engine

Edytor UE5: zwany Unreal Editor: posiada analogiczną do Unity strukturę panelów, choć używa własnego nazewnictwa. Rozpoznanie tych odpowiedników jest kluczowym krokiem transferu wiedzy:

Panel Unity	Odpowiednik w UE5	Funkcja
Scene View	Viewport	Interaktywny widok 3D świata gry / aplikacji
Game View	PIE (Play In Editor)	Symulacja aplikacji: uruchamiana przyciskiem Play
Hierarchy	Outliner	Lista wszystkich aktorów w aktywnym poziomie
Inspector	Details Panel	Podgląd i edycja komponentów zaznaczonego aktora
Project	Content Browser	Hierarchia zasobów (assets) projektu
Console	Output Log	Logi, ostrzeżenia i błędy silnika
Package Manager	Plugins	Zarządzanie wtyczkami silnika
Prefab	Blueprint Class	Wielokrotnie użyteczny szablon aktora ze łogiką
C# Script	Blueprint / C++	Logika aplikacji: wizualna (BP) lub tekstowa (C++)



Rys. 1. Interfejs edytora Unreal Engine 5 z opisanymi panelami

2.3. Kluczowe różnice między Unity, a Unreal Engine

Znajomość różnic projektowych między silnikami jest niezbędna do podejmowania świadomych decyzji technologicznych. Poniższa tabela syntetyzuje najważniejsze z nich:

Aspekt	Unity	Unreal Engine 5
Język skryptowania	C# (MonoBehaviour)	Blueprints (vizualny) lub C++ (natywny)
Jednostki świata	1 jednostka = 1 metr (konwencja)	1 jednostka = 1 centymetr (!): świat w cm
System assetów	Lupy pliki na dysku, import ręczny	Wewnętrzna baza .uasset, automatyczne przetwarzanie
Renderowanie (domyślne)	URP lub HDRP (własne pipeline)	Deferred Rendering + Lumen + Nanite
Oświetlenie	Baked (pieczenie) lub real-time	Real-time Lumen lub baked Lightmass

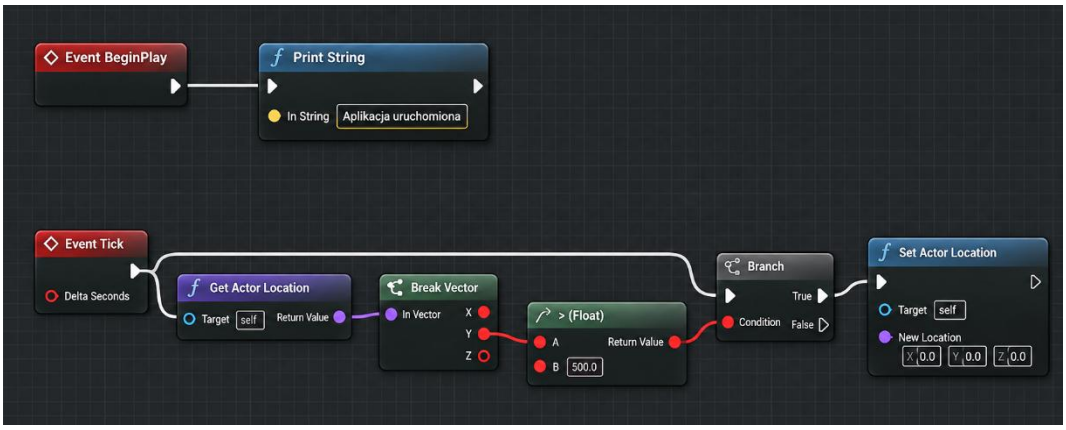
System cząstek	VFX Graph / Particle System	Niagara (własny, bardzo rozbudowany)
Teren i roślinność	Terrain + SpeedTree (przez asset)	Landscape + PCG + Foliage Tool (wbudowane)
Wieloosobowe	Netcode for GameObjects / Relay	Wbudowany Replication Graph (dojrzały)
Licencja	Personal (darmowy do $100k/rok)	Darmowy; 5% royalty po przekroczeniu $1M przychodu
Dokumentacja i społ., w PL	Bardzo szeroka, dużo materiałów w PL	Obszerne EN, mniej PL zasobów

Jednostki świata w Unreal Engine: pułapka dla użytkowników Unity
Unreal Engine używa centymetrów jako jednostki bazowej. Oznacza to, że obiekt o rozmiarze 180 cm (postać ludzka) będzie miał Scale = (1, 1, 1), ale jego wymiar w siatce to 180 jednostek. Importując modele z Unity lub Maya (gdzie 1 j. = 1 m), należy przeskalować je przez współczynnik 100. RealityScan i większość exporterów FBX obsługuje to automatycznie: należy jednak zawsze weryfikować skalę po imporcie.

2.4. System Blueprints: wizualne programowanie

Blueprints Visual Scripting (w skrócie: BP) to graficzny język programowania wbudowany w Unreal Engine. Zamiast pisania kodu w IDE, programista **przeciaga i łączy węzły (nodes)** reprezentujące operacje, zmienne i zdarzenia. Każde połączenie między węzłami jest jawnym przepływem danych lub sterowania.

Typ węzła	Opis	Odpowiednik w C#
Event Node	Punkt startowy wykonania (np. Event BeginPlay, Event Tick)	void Start() / void Update()
Function Node	Wywołanie funkcji lub metody (np. Set Actor Location)	obiekt.MetodaPubliczna()
Variable Node	Pobranie (Get) lub ustawienie (Set) wartości zmiennej	zmienna / zmienna = wartość
Branch	Rozgałęzienie warunkowe (if-else)	if (warunek) { ... } else { ... }
For Loop	Pętla z licznikiem	for (int i = 0; i < N; i++) { ... }
Cast To	Rzutowanie na typ (sprawdzenie klasy aktora)	(TypDocelowy)obiekt
Print String	Wypisanie tekstu do ekranu (debug)	Debug.Log(text)
Timeline	Animacja wartości w czasie (krzywa)	Coroutine / AnimationCurve



Rys. 2. Przykładowy graf Blueprints: prosta logika interakcji

Blueprints są kompilowane do bytecodu wykonywanego przez wirtualną maszynę UE: są zatem wolniejsze niż natywny kod C++, ale różnica ta jest zauważalna dopiero przy bardzo intensywnych obliczeniach (tysiące wywołań na klatkę). Dla przetwarzania logiki aplikacji i interakcji użytkownika Blueprints są absolutnie wystarczające.

2.5. Lumen i Nanite: rewolucja renderowania w UE5

Nanite to system wirtualizacji geometrii pozwalający na używanie modeli o dziesiątkach milionów polygonów bez ręcznego tworzenia LOD. [3] Silnik automatycznie wybiera poziom szczegółowości i renderuje tylko te części siatki, które są widoczne z aktualnego punktu kamery, z uwzględnieniem projekcji perspektywicznej.

Lumen to system dynamicznego globalnego oświetlenia obliczający w czasie rzeczywistym odbić, rozproszenie i światło otoczenia (ambient occlusion) bez konieczności wcześniejszego „pieczenia” (baking) światła. Dzięki Lumen zmiana położenia słońca lub włączenie lampy natychmiast aktualizuje oświetlenie całej sceny.

> **Nanite i Lumen**
> Nanite i Lumen wymagają karty graficznej obsługującej DirectX 12 (NVIDIA GTX 1080 lub nowsza, AMD RX 5700 lub nowsza). Na starszym sprzęcie należy w Project Settings wyłączyć Lumen i zastosować tradycyjne baked lighting. Nanite można wyłączyć per-mesh w Details Panel → Nanite Settings.

2.6. Unreal Engine poza branżą gier

Termin **"industry-agnostic real-time 3D"** opisuje kierunek, w którym od kilku lat zmierzają obaj wiodący producenci silników. Poniższe zestawienie ilustruje zakres zastosowań UE poza sektorem rozrywkowym:

Branża	Zastosowanie UE	Przykład
Film i TV	Wirtualna produkcja (LED wall + UE)	The Mandalorian, House of the Dragon
Architektura	Interaktywne wizualizacje przedrealizacyjne	Zaha Hadid Architects, BIG Architects
Motoryzacja	Konfiguratory pojazdów, symulacje aerodynamiki	BMW iVision Circular, Mercedes-Benz
Szkolenia	Symulatory dla wojska, medycyny, lo tnictwa	L3Harris, Boeing, U.S. Army
Inżynieria	Wizualizacje projektów przemysłowych	Siemens, GE Aviation, Shell

Edukacja	Wirtualne laboratoria i muzea	Smithsonian Institution, NASA
Broadcast / Sport	Wirtualne studia, analiza taktyczna	Sky Sports, ESPN, TVN

3. Przygotowanie środowiska

Epic Games Launcher	Pobierz i zainstaluj Epic Games Launcher ze strony unrealengine.com. Konto Epic Games wymagane (bezpłatne).
Wersja UE	Zainstaluj Unreal Engine 5.3 lub nowszy przez zakładkę Unreal Engine → Library → + (Install Engine). Uwaga: instalacja zajmuje ok. 60–80 GB.
Sprzęt	Min. GPU: NVIDIA GTX 1080 / AMD RX 5700. RAM: 16 GB (zalecane 32 GB). Dysk SSD silnie zalecany: UE intensywnie korzysta z I/O.
Modele 3D	Przygotuj co najmniej trzy pliki .fbx z poprzednich zajęć (Lab 2 lub Lab 5) lub pobierz nowe z Fab.com (Marketplace Epic Games).

Fab.com: nowy Marketplace Epic Games
W 2024 roku Epic Games połączył Unreal Engine Marketplace, Sketchfab Store i ArtStation Marketplace w jedną platformę: fab.com. Zawiera tysiące darmowych i płatnych zasobów kompatybilnych z Unreal Engine (filtry: Free, Compatible with UE5).

4. Ćwiczenia praktyczne

4.1. Tworzenie projektu i konfiguracja środowiska

Pierwsze uruchomienie Unreal Engine może być przytłaczające: interfejs jest bogatszy niż Unity, a konwencje różnią się od znanych. Celem tego kroku jest opanowanie nawigacji i przygotowanie czystego środowiska pracy.

Krok 1: Tworzenie projektu

1. Otwieramy Epic Games Launcher → Unreal Engine → Launch.
2. W Unreal Project Browser wybieramy:
   - Kategoria: Games (uproszczone ustawienia domyślne) lub Architecture, Engineering & Construction (jeśli planujemy wizualizację).
   - Szablon: Blank (pusty projekt: najczystszy punkt startowy).
   - Raytracing: OFF (chyba że GPU obsługuje raytracing sprzętowy).
   - Quality: Maximum.
   - Starter Content: YES (gotowe materiały i modele do ćwiczeń).
3. Nadajemy projekt nazwę Lab06_UnrealIntro i wybieramy folder docelowy.
4. Klikamy Create. Pierwsze uruchomienie trwa kilka minut: UE kompiluje shadery.

Krok 2: Orientacja w edytorze

Po załadowaniu projektu widzimy domyślny poziom (Level). Wykonujemy podstawowe ćwiczenia nawigacyjne:

1. **Viewport:** PPM (prawy przycisk myszy) + W/A/S/D: lot kamerą FPS. Koło myszy: prędkość lotu (scroll). Alt + LPM: orbita wokół zaznaczonego obiektu.
2. **Dodawanie obiektów:** Toolbar → przycisk Add (+) lub Shift+A. Można również przeciągnąć asset z Content Browser do Viewport.
3. **Zaznaczanie:** LPM w Viewport lub kliknięcie w Outliner. Shift: zaznaczenie wielokrotne.
4. **Transformacje:** W: Move, E: Rotate, R: Scale (identyczne jak w Unity).

> **Zapisywanie w UE: różnica względem Unity**
> W Unreal Engine poziom (Level) zapisuje się przez Ctrl+S lub File → Save Current Level. Projekt sam w sobie nie ma pliku sceny: każdy poziom to osobny plik .umap w folderze Content. Niezapisany poziom oznaczony jest gwiazdką (*) w tytule okna.

4.2. Budowa środowiska interaktywnego

Tworzymy scenę 3D składającą się z podłoża, oświetlenia i zaimportowanych modeli. Celem nie jest fotorealizm, lecz poprawna konfiguracja środowiska i poznanie narzędzi UE.

Krok 1: Tworzenie podłoża i środowiska

1. Usuwamy wszystkie domyślne obiekty ze sceny: Ctrl+A (zaznacz wszystko) → Delete.
2. Dodajemy podłoże: Add (+) → Shapes → Plane. W Details Panel: Scale (20, 20, 1), Location (0, 0, 0).
3. Dodajemy środowisko atmosferyczne: Add (+) → Visual Effects → Sky Atmosphere. UE doda automatycznie obłoki i gradient nieba.
4. Dodajemy oświetlenie słoneczne: Add (+) → Lights → Directional Light. Rotation: (-45, -30, 0). Intensity: 10 lux.
5. W Details Panel nowego Directional Light zaznaczamy
   - **Atmosphere Sun Light: ON**: światło słoneczne automatycznie wpływa na Sky Atmosphere.
6. Dodajemy Exponential Height Fog: Add (+) → Visual Effects → Exponential Height Fog (opcjonalne, ale poprawia realizm głębi).

Krok 2: Konfiguracja kamery

1. Dodajemy aktora CameraActor: Add (+) → Cinematics → Camera Actor. Ustawiamy Location i Rotation w Details Panel tak, by obejmował całą scenę.
2. Aby podglądnąć widok kamery w Viewport: zaznaczamy Camera Actor → w Viewport klikamy ikonę aparatu fotograficznego (Look Through Camera). Wyjdźmy przez Ejection (klawisz G).
3. Alternatywnie: Viewport → Perspectives → Camera Actor: przełącza widok na wybrany aktor kamery.

4.3. Import modeli 3D do Unreal Engine

Import do UE różni się od Unity: UE konwertuje pliki zewnętrzne do własnego formatu `.uasset` i przechowuje je wewnątrz bazy danych projektu w folderze `Content/`. Oryginalny plik .fbx można po imporcie usunąć: UE nie potrzebuje go do dalszej pracy.

Krok 1: Import pliku FBX

1. W Content Browser przejdź do folderu Content. Kliknij prawym przyciskiem → New Folder i utwórz folder Models.
2. Przeciągnij plik .fbx z Eksploratora plików systemu operacyjnego bezpośrednio na folder Models w Content Browser.
3. UE wyświetli okno FBX Import Options. Konfigurujemy:
   - **Import Mesh: ON** : importujemy geometrię.
   - **Auto Generate Collision: ON**: UE wygeneruje przybliżony Collider.
   - **Import Materials: ON**: UE spróbuje odwzorować materiały z pliku FBX.
   - **Import Textures: ON**: tekstury z pliku FBX zostaną zaimportowane jako osobne .uasset.
   - **Transform: Import Uniform Scale: 1.0** (korygujemy jeśli model jest źle przeskalowany). [7]
4. Klikamy Import All.
5. Po imporcie w Content Browser pojawią się pliki: Static Mesh (SM_), materiał (M_), tekstura (T_). To konwencja nazewnicza UE.

Krok 2: Weryfikacja i konfiguracja importu

1. Dwukrotnie klikamy na Static Mesh: otworzy się Static Mesh Editor. Widoczne są:
   - Podgląd modelu z optymalną kamerą,
   - Liczba triangle (Tris) i vertexów (Verts): w prawym dolnym rogu podglądu,
   - Automatycznie wygenerowane LOD (LOD0, LOD1...): w zakładce LOD Settings,
   - Kolizja: można ją wizualizować przez Collision → Show Simple Collision.
2. Zamykamy Static Mesh Editor. Przeciagamy model z Content Browser do Viewport.
3. Powtarzamy dla pozostałych modeli, rozmieszczając je w scenie.

4.4. Tworzenie logiki z Blueprints

Blueprints są sercem programowania w UE dla osób niespecjalizujących się w C++. Stworzymy dwie Blueprinty: jeden do obsługi interaktywnego obiektu (zmiana koloru po zbliżeniu) i jeden jako kontroler globalnej logiki sceny.

Krok 1: Blueprint klasy: interaktywny obiekt

1. W Content Browser: prawym przyciskiem → Blueprint Class. W oknie wyboru klasy bazowej wybieramy Actor. Nadajemy nazwę BP_InteraktywnyObiekt.
2. Dwukrotnie klikamy BP_InteraktywnyObiekt: otwiera się Blueprint Editor. Składa się z:
   - **Viewport:** wizualny podgląd aktora i jego komponentów.
   - **Event Graph:** główne płótno do tworzenia logiki przez węzły.
   - **My Blueprint (panel lewy):** zmienne, funkcje, komponenty aktora.
3. W sekcji Components (lewy panel) klikamy Add → Static Mesh Component. Przypisujemy jeden z zaimportowanych modeli do właściwości Static Mesh.
4. Dodajemy zmienną: w My Blueprint → Variables → +. Typ: Linear Color. Nazwa: KolorPodswietlenia. Wartość domyślna: żółty (1, 1, 0, 1).
5. Budujemy graf w Event Graph:

```
Event BeginPlay
  ↳▶ Print String ("BP_InteraktywnyObiekt uruchomiony")

Event ActorBeginOverlap (called when another actor enters this actor's trigger)
```

```
   ↳ Set Material (Target: StaticMeshComponent, Element Index: 0,
        Material: dynamicMat z węzła Create Dynamic Material Instance)
      ↳ Set Vector Parameter Value (ParameterName: BaseColor,
           Value: GET KolorPodswietlenia)

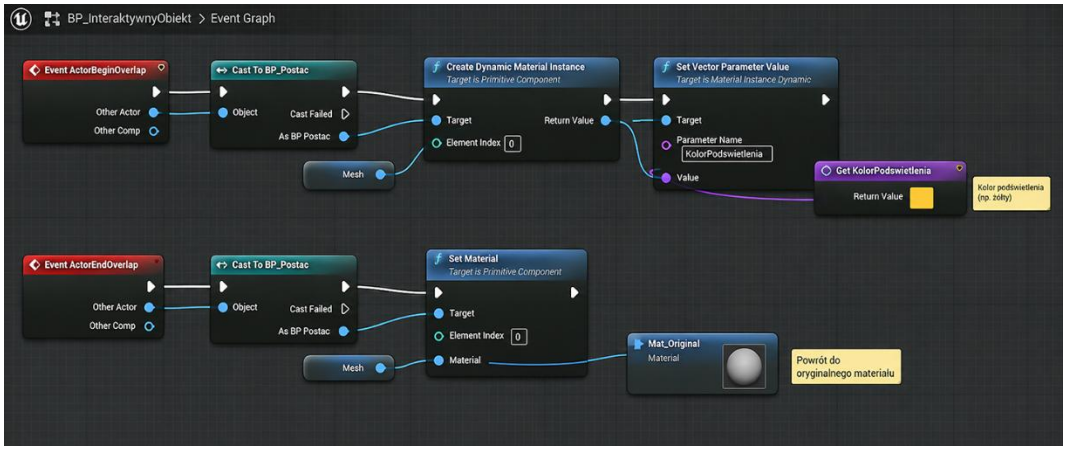
Event ActorEndOverlap
   ↳ Set Material (powrót do oryginalnego materiału)
```

Dynamiczne materiały w Blueprints

Aby zmieniać kolor materiału przez Blueprints, musimy najpierw stworzyć dynamiczną instancję materiału przez węzeł Create Dynamic Material Instance. Dopiero na instancji możemy wywołać Set Scalar Parameter Value lub Set Vector Parameter Value. Analogia do Unity: odpowiednik material.color = ... na renderer.material (nie sharedMaterial).

Krok 2: Dodanie Trigger Box

1. W Blueprint Editor → Viewport → Add → Box Collision. Skalujemy Box tak, by obejmował obiekt z marginesem (np. Scale 2.0 × 2.0 × 2.0).
2. Zaznaczamy Box Collision → Details → sekcja Collision: Collision Presets = Overlap All Dynamic.
3. W Event Graph klikamy PPM na wolnej przestrzeni → szukamy „On Component Begin Overlap" dla Box Collision: dodajemy odpowiednie zdarzenia.
4. Kompilujemy Blueprint przyciskiem Compile (górna belka). Pojawienie się zielonej ikony ✓ oznacza brak błędów.
5. Przeciagamy BP_InteraktywnyObiekt z Content Browser do sceny. Klikamy Play (Alt+P) i wchodzimy w strefę aktora: obs erwujemy zmianę koloru.



Rys, 3. Graf Blueprints: logika interakcji z obiektem

4.5. Podstawowa interakcja użytkownika

UE5 używa systemu **Enhanced Input** jako domyślnego systemu wejścia od wersji 5.1. Analogicznie do nowego Input System w Unity, Enhanced Input opiera się na **Input Actions** i **Input Mapping Contexts**: plikach konfiguracyjnych zamiast na bezpośrednim sprawdzaniu klawiszy w kodzie. [7]

Krok 1: Konfiguracja Input Action

1. W Content Browser → Create → Input → Input Action. Nadajemy nazwę IA_Aktywuj. Value Type: Digital (bool): akcja binarna (wciśnięty / nie).
2. Tworzymy Input Mapping Context: Content → Create → Input → Input Mapping Context. Nazwa: IMC_Domyslny.

3. Dwukrotnie klikamy IMC_Domyslny → +Mappings → wybieramy IA_Aktywuj. Pod mapowaniem klikamy + i przypisujemy klawisz E.

Krok 2: Obsługa wejścia w Blueprint gracza

1. Tworzymy nowy Blueprint: Create → Blueprint Class → Pawn (lub Character). Nazwa: BP_Gracz.
2. W Event Graph dodajemy węzeł Enhanced Input Action (szukamy "IA_Aktywuj"): generuje zdarzenia Triggered, Started, Completed.
3. Do węzła Triggered podłączamy Print String z tekstem „Aktywowano!”.
4. W Event BeginPlay dodajemy:
   - `Get Player Controller` → `Add Mapping Context` (Input: IMC_Domyslny, Priority: 0)
5. W Project Settings → Maps & Modes ustawiamy Default Pawn Class na BP_Gracz.
6. Testujemy: Play i naciskamy E: komunikat powinien pojawić się na ekranie.

4.6. Porównanie pipeline’u Unity i Unreal Engine

Ostatnim ćwiczeniem jest systematyczna analiza porównawcza obu środowisk. Na podstawie doświadczeń z sześciu laboratoriów i bieżących zajęć uzupełniamy dokument w folderze Documentation.

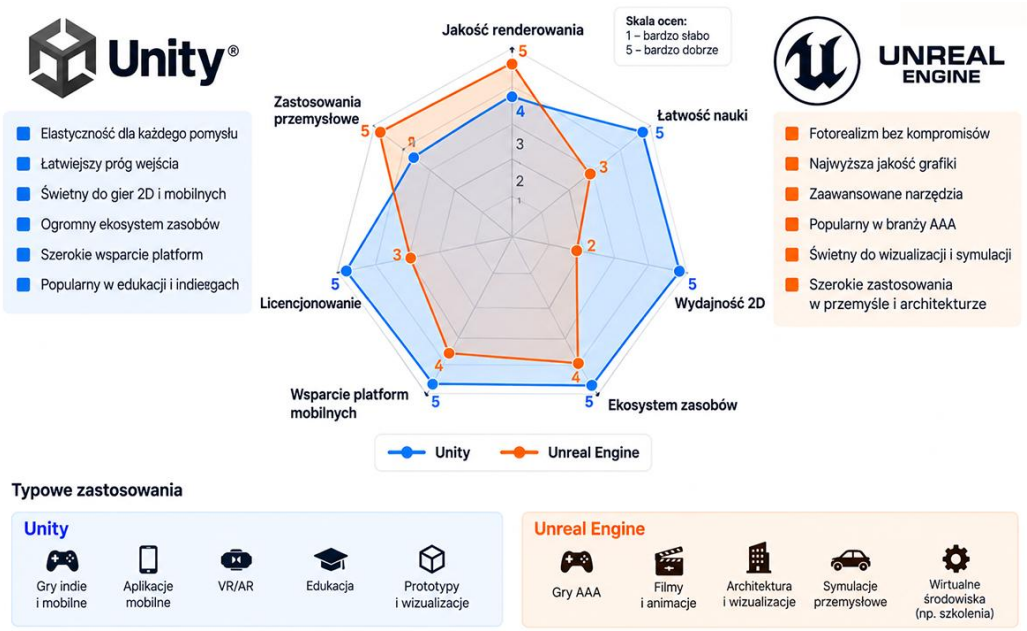
Krok 1: Porównanie konkretnych operacji

Dla każdej z poniższych czynności notujemy, jak ćwiczenie przebiegało w Unity (Lab 1-5) i jak przebiegało w UE (Lab 6), oraz oceniamy trudność w skali 1–5:

Czynność	Unity (sposób / ocena)	Unreal Engine (sposób / ocena)
Tworzenie projektu	(opis)	(opis)
Import modelu FBX	(opis)	(opis)
Konfiguracja materiału	(opis)	(opis)
Dodanie oświetlenia	(opis)	(opis)
Prosta logika interakcji	(skrypt C#)	(Blueprint)
Obsługa wejścia użytkownika	(Input.GetAxis)	(Enhanced Input)
Debugowanie	(Debug.Log + Console)	(Print String + Output Log)
Czas inicjalizacji projektu	(minuty)	(minuty)

Krok 2: Argumentacja wyboru silnika

Na podstawie analizy sformułujemy wniosek: dla jakiego typu projektów wybralibyśmy Unity, a dla jakiego Unreal Engine. Odpowiedź uzasadniamy co najmniej trzema konkretnymi argumentami technicznymi lub organizacyjnymi.



Rys. 4. Zestawienie cech Unity vs Unreal Engine:

5. Lista kontrolna wykonanych zadań

Przed zakończeniem laboratorium sprawdź, czy wszystkie poniższe elementy zostały zrealizowane:

☐	Projekt Unreal Engine 5 utworzony z szablonem Blank, folder Lab06_UnrealIntro.
☐	Scena zawiera podłoże (Plane), Sky Atmosphere i skonfigurowane Directional Light.
☐	Kamera (Camera Actor) ustaw iona i przetestowana przez Look Through Camera.
☐	Co najmniej trzy modele FBX zaimportowane do Content Browser jako .uasset.
☐	Wszystkie modele poprawnie skalowane i rozmieszczone w scenie.
☐	Blueprint BP_InteraktywnyObiekt zawiera Box Trigger i zmieni kolor obiektu przy wejściu.
☐	Blueprint skompilowany bez błędów (zielona ikona Compile).
☐	Input Action IA_Aktywuj i Input Mapping Context IMC_Domyslny skonfigurowane.
☐	Klawisz E wywołuje akcję i wyświetla komunikat w PIE (Play In Editor).
☐	Tabela porównania Unity / UE uzupełniona z ocenami w Documentation.
☐	Dokument Sprawozdanie.md z argumentacją wyboru silnika zapisany w Documentation.

6. Literatura i odwołania

6.1. Pozycje podstawowe

[1] Hocking, J. (2018). Unity in Action: Multiplatform Game Development in C#. Manning Publications. ISBN 978-1617294969. Rozdział 1: Porównanie środowisk deweloperskich gier.

[2] Sweeney, T. (2023). The State of Unreal 2023: GDC Keynote. Epic Games. Dostępne: https://www.gdcvault.com/play/1029049 (dostęp: maj 2026).

[3] Epic Games. (2024). Unreal Engine 5 Documentation: Overview. https://docs.unrealengine.com/5.3/en-US/unreal-engine-5-migration-guide/

[4] Downie, M., Taylor, P. (2022). Real-Time Filmmaking with Unreal Engine. Focal Press, Routledge. ISBN 978-1032153742.

[5] Sherrod, A., Jones, W. (2013). Beginning Unreal Game Development. Apress. ISBN 978-1484256688.

6.2. Dokumentacja techniczna

[6] Epic Games. (2024). Blueprints Visual Scripting. https://docs.unrealengine.com/5.3/en-US/blueprints-visual-scripting-in-unreal-engine/ (dostęp: maj 2026)

[7] Epic Games. (2024). Enhanced Input Plugin. https://docs.unrealengine.com/5.3/en-US/enhanced-input-in-unreal-engine/ (dostęp: maj 2026)

[8] Epic Games. (2024). Nanite Virtualized Geometry. https://docs.unrealengine.com/5.3/en-US/nanite-virtualized-geometry-in-unreal-engine/ (dostęp: maj 2026)

[9] Epic Games. (2024). Lumen Global Illumination and Reflections. https://docs.unrealengine.com/5.3/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/ (dostęp: maj 2026)

6.3. Materiały uzupełniające

Zasób	Opis	URL / źródło
Unreal Online Learning	Oficjalne kursy Blueprints i UE5 od podstaw (darmowe)	dev.epicgames.com/community
Fab.com	Marketplace Epic Games: darmowe i płatne zasoby dla UE5	fab.com
Unreal Sensei (YouTube)	Tutoriale UE5 dla początkujących: interfejs, Blueprints, oświetlenie	youtube.com/@UnrealSensei
Brackeys (YouTube)	Popularne tutoriale Unity; autor ogłosił powrót z materiałami UE5	youtube.com/@Brackeys
Game Programming Patterns: online	Wzorce projektowe przekładalne na Blueprints	gameprogrammingpatterns.com