Instrukcja laboratoryjna nr 3

Interakcja użytkownika, sterowanie i interfejs aplikacji

1. Cel i kontekst ćwiczenia

Dotąd budowaliśmy sceny, które działały „same z siebie”: obiekty obracały się, ale użytkownik nie miał żadnego wpływu na to, co się dzieje. Aplikacja interaktywna różni się od animacji właśnie tym, że reaguje na działania człowieka w czasie rzeczywistym. To sprzężenie zwrotne między użytkownikiem a systemem nazywamy interakcją, a jego jakość decyduje o tym, czy aplikacja jest odczuwana jako responsywna i przyjemna w użytkowaniu.

W trakcie tego laboratorium zaprojektujemy i zaimplementujemy trójwarstwowy system: wejście (ang. input) → logika → wyjście (ang. output), który jest fundamentem każdej aplikacji interaktywnej, od prostej gry zręcznościowej po przemysłowy pulpit operatora.

Po zakończeniu tego laboratorium student

1. rozumie różnicę między starym Input Managerem a nowym Input System i wie, którego używać,
2. potrafi odczytywać klawiaturę, mysz i osę analogową i reagować na nie w skrypcie C#,
3. rozumie mechanizm raycastingu i potrafi wybrierać obiekty sceny kliknięciem myszy,
4. potrafi zbudować prosty interfejs użytkownika z użyciem Canvas, TextMeshPro i Button,
5. potrafi połączyć elementy UI z logiką aplikacji przez zdarzenia (Events),
6. rozumie podstawowe prawa UX i potrafi zanalizować jakość interakcji swojej aplikacji.

2. Wprowadzenie teoretyczne

2.1. Architektura systemu wejścia w Unity

Unity oferuje dwa różne systemy obsługi wejścia użytkownika, współistniejące w bieżących wersjach edytora:

Właściwość	Input Manager (stary)	Input System (nowy, pakiet)
API	Input.GetKey(), Input.GetAxis()	InputAction, PlayerInput komponent
Konfiguracja	Edit → Project Settings → Input	Plik .inputactions (graficzny edytor)
Urządzenia	Klawiatura, mysz, joystick	Klawiatura, mysz, gamepad, touch, XR
Rebinding	Brak wbudowany	Obsługiwany natywnie
Zalecenie	Prostsze projekty / nauka	Projekty produkcyjne, multi-platform

Na tych zajęciach używamy klasycznego Input Managera, który jest dostępny bez instalacji dodatkowych pakietów i jest wystarczający do opanowania koncepcji. W projekcie zaliczeniowym można wybrać dowolny z systemów.

2.2. Osie i przyciski: model abstrakcji wejścia

Dobry system wejścia działa na poziomie abstrakcji, nie konkretnych klawiszy. Zamiast sprawdzać „czy wciśnięto W”, sprawdzamy „jaka jest wartość osi Vertical”. Dzięki temu ten sam kod obsługuje klawiaturę, kontroler i joystick bez modyfikacji.

Obieś / przycisk	Klawisze domyślne	Zakres wartości	Zastosowanie
Horizontal	A/D, strzałki lewo/prawo, joystick X	-1.0 … +1.0	Ruch boczny, obrót kamery
Vertical	W/S, strzałki góra/dół, joystick Y	-1.0 … +1.0	Ruch przód/tył
Mouse X	Ruch myszy poziomo	delta pikseli/klatkę	Obrót kamery FPS
Mouse Y	Ruch myszy pionowo	delta pikseli/klatkę	Nachylenie kamery FPS
Fire1	Lewy Ctrl, LMB	0 lub 1 (bool)	Akcja główna, strzał
Jump	Spacja	0 lub 1 (bool)	Skok, potwierdzenie

2.3. Raycasting: rzut promienia w scenie 3D

Raycasting (rzut promienia) to technika polegająca na wysłaniu niewidocznego promienia z danego punktu w danym kierunku i sprawdzeniu, z jakim obiektem się zderzy. Jest to podstawowy mechanizm wykrywania obiektów pod kursorem myszy, co leży u podstaw każdego systemu wskazywania i klikania w środowiskach 3D.

W Unity przeliczamy pozycję kursora na ekranie na promień w przestrzeni 3D przy użyciu metody kamery:

```
// Przeliczenie pozycji kursora (2D) na promień w świecie 3D
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

// Rzut promienia: wynik trafia do zmiennej 'hit'
if (Physics.Raycast(ray, out RaycastHit hit, maxDistance: 100f))
{
    // hit.collider.gameObject: obiekt, który został trafiony
    // hit.point          : dokładny punkt kontaktu w przestrzeni 3D
    // hit.normal         : normalna powierzchni w punkcie kontaktu
    Debug.Log("Trafiono: " + hit.collider.gameObject.name);
}
```

Kluczowym warunkiem działania raycastingu jest obecność komponentu Collider na docelowym obiekcie. Bez kolajdera promień przelatuje przez obiekt, nie zwracając żadnego trafienia: to jedna z najczęstszych przyczyn błędów początkujących.

2.4. Komunikacja między komponentami

Skrypty w Unity rzadko działają w izolacji: zazwyczaj jeden skrypt musi modyfikować stan innego. Istnieją trzy główne wzorce komunikacji:

Wzorzec	Mechanizm Unity	Zalety	Wady
Bezpośrednio przez referencję	GetComponent<T>(), publiczne pole	Prosta, szybka	Silne powiązanie (tight coupling)
Przez zdarzenia (Events)	UnityEvent, C# event/delegate	Słabe powiązanie, elastyczna	Trudniejsza do debugowania
Przez wzorzec Singleton	Statyczna instancja klasy Manager	Globalny dostęp	Trudna do testowania

Na tych zajęciach używamy przede wszystkim referencji bezpośrednich (przez GetComponent) oraz zdarzeń UnityEvent (do UI). Wzorzec Singleton zostanie omówiony szerzej w Laboratorium 4.

2.5. System UI Unity: Canvas i TextMeshPro

Interfejs użytkownika w Unity oparty jest na hierarchii obiektów zakotwiczonych w komponencie Canvas. Canvas jest płotnem, na którym rysowane są wszystkie elementy 2D: przyciski, etykiety, obrazy i paski postępu.

Tryb renderowania Canvas	Opis	Kiedy używać
Screen Space – Overlay	UI renderowane na wierzchu wszystkiego, po scenie 3D	HUD, menu, liczniki: domyślny tryb
Screen Space – Camera	UI przypisane do kamery, może być przesłonięte przez obiekty 3D	Efekty depth of field na UI
World Space	Canvas jest obiektem w przestrzeni 3D sceny	Tablice informacyjne w grze, VR

TextMeshPro (TMP) to zaawansowany system renderowania tekstu, domyślnie wbudowany w Unity. W porównaniu do starego komponentu Text zapewnia ostre renderowanie w każdej rozdzielczości, obsługę bogatego formatowania i znacznie większą elastyczność. Zawsze należy stosować TMP zamiast starszego Text.

2.6. Podstawy UX: dlaczego interakcja musi być projektowana

UX (User Experience) to całościowe doświadczenie użytkownika obcującego z produktem. W aplikacjach interaktywnych zły UX objawia się m.in. brakiem feedbacku (użytkownik nie wie, czy kliknięcie zadziałało), nieczytelnym interfejsem lub nieintuicyjnym sterowaniem. Don Norman w „The Design of Everyday Things”: książce z listy literatury kursu: opisuje sześć zasad dobrego projektowania. Cztery z nich bezpośrednio przekładają się na interakcję w grach i aplikacjach:

- **Widoczność** (Visibility): elementy interaktywne powinny być wyraźnie odznaczać się od tła.
- **Feedback**: każda akcja użytkownika powinna natychmiast wywoływać zauważalną odpowiedź systemu.
- **Mapowanie** (Mapping): sterowanie powinno odpowiadać intuicyjnym oczekiwaniom (W oznacza w przód, nie w tył).
- **Ograniczenia** (Constraints): niedostępne opcje powinny być zablokowane lub ukryte, nie po prostu nic nie robić.

3. Przygotowanie do ćwiczenia

Projekt bazowy	Kontynuujemy projekt z Lab 1 lub Lab 2. Tworzymy nową scenę ScenaLab03 w folderze _Scenes.
TextMeshPro	Przy pierwszym użyciu TMP Unity wyświetli okno importu zasobów: klikamy Import TMP Essentials.
Scena robocza	Scena powinna zawierać płaską podłogę (Plane), kamerę i światło. Dodajemy minimum 3 obiekty 3D z Colliderami.

4. Ćwiczenia praktyczne

4.1. Sterowanie obiektem z klawiatury

Pierwszym krokiem jest implementacja najbasrdziej fundamentalnego mechanizmu interakcji: przemieszczania obiektu przez użytkownika. Zbudujemy skrypt sterowania, który będzie poprawnie uwzględniał czas między klatkami, dzięki czemu prędkość ruchu będzie taka sama niezależnie od liczby FPS.

Krok 1: Skrypt SterowanieObiektem

W folderze Assets/Scripts tworzymy nowy skrypt C# o nazwie SterowanieObiektem i wypelniamy go poniższą treścią:

```
using UnityEngine;

public class SterowanieObiektem : MonoBehaviour
{
    [SerializeField] private float predkosc     = 5f;
    [SerializeField] private float predkoscObrotu = 120f;

    void Update()
    {
        // Odczyt osi: wartość w zakresie -1 … +1
        float poziom    = Input.GetAxis("Horizontal");  // A/D
        float pionowy   = Input.GetAxis("Vertical");    // W/S

        // Ruch w lokalnym układzie współrzędnych obiektu
        Vector3 ruch = new Vector3(poziom, 0f, pionowy);
        transform.Translate(ruch * predkosc * Time.deltaTime, Space.World);

        // Obrót obiektu klawiszami Q/E
        if (Input.GetKey(KeyCode.Q))
            transform.Rotate(0f, -predkoscObrotu * Time.deltaTime, 0f);
        if (Input.GetKey(KeyCode.E))
            transform.Rotate(0f,  predkoscObrotu * Time.deltaTime, 0f);
    }
}
```

Metoda `Input.GetAxis()` zwraca płynną, interpolowaną wartość osi (analog), w przeciwieństwie do `Input.GetKey()`, która zwraca skokowy sygnał boolowski. W sterowaniu postacią zawsze preferujemy wersję analogową: daje naturalniejszy ruch (przyspieszanie i zwalnianie).

1. Przyłączamy skrypt do wybranego obiektu 3D w scenie (np. Cube).
2. Uruchamiamy scenę i sprawdzamy sterowanie klawiszami W/A/S/D oraz Q/E.
3. Eksperymentujemy z wartością Prędkość w Inspectorze w trakcie działania aplikacji.

Krok 2: Wariant: sterowanie kamerą FPS

Alternatywnie do sterowania obiektem możemy zastosować klasyczne sterowanie kamerą FPS, gdzie mysz obraca widokiem. Poniższy skrypt należy przyłączyć do obiektu kamery, nie do innego obiektu:

```
using UnityEngine;

public class SterowanieKamera : MonoBehaviour {
    [SerializeField] private float czulosc = 2f;
    [SerializeField] private float predkosc = 5f;
    private float rotacjaX = 0f;   // nachylenie w górę/dół
```

```
    void Start() {
        // Ukrycie i zablokowanie kursora w oknie aplikacji
        Cursor.lockState = CursorLockMode.Locked;
    }

    void Update() {
        // Obrót poziomy: cały obiekt kamery
        float myszX = Input.GetAxis("Mouse X") * czulosc;
        transform.Rotate(0f, myszX, 0f);

        // Pochylenie pionowe: ograniczone do -80..+80 stopni
        float myszY = Input.GetAxis("Mouse Y") * czulosc;
        rotacjaX  -= myszY;
        rotacjaX   = Mathf.Clamp(rotacjaX, -80f, 80f);
        transform.localEulerAngles = new Vector3(rotacjaX,
            transform.localEulerAngles.y, 0f);

        // Ruch WASD
        float poziom  = Input.GetAxis("Horizontal");
        float pionowy = Input.GetAxis("Vertical");
        transform.Translate(new Vector3(poziom, 0f, pionowy)
            * predkosc * Time.deltaTime);
    }
}
```

Uwaga: odblokowywanie kursora

Aby wyjść z trybu zablokowanego kursora podczas testowania, naciśnij klawisz Escape. Cursor.lockState = CursorLockMode.None; należy dodać w warunku obsługi klawisza Escape w gotowej aplikacji.

4.2. Raycasting: wybieranie obiektów kliknięciem

Zaimplementujemy mechanikę pozwalającą użytkownikowi klikać na obiekty sceny lewym przyciskiem myszy. Po kliknięciu wybrany obiekt zmieni kolor, co będzie pierwszym przykładem feedbacku wizualnego.

Krok 1: Skrypt WyborObiektu

Tworzymy nowy skrypt C# o nazwie WyborObiektu i przyłączamy go do pustego obiektu GameManager w hierarchii (Hierarchy → Create Empty, nazwa: GameManager):

```
using UnityEngine;
public class WyborObiektu : MonoBehaviour {
    [SerializeField] private Color kolorWyboru = Color.yellow;
    private GameObject wybranyObiekt;
    private Color oryginalnyKolor;

    void Update() {
        // Sprawdzamy lewy przycisk myszy
        if (!Input.GetMouseButtonDown(0)) return;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        if (Physics.Raycast(ray, out RaycastHit hit, 100f)) {
            PrzywrocKolor();   // zdejmij podświetlenie z poprzedniego
            wybranyObiekt = hit.collider.gameObject;
            PodswietlObiekt(); // podświetl nowy

            Debug.Log("Kliknięto: " + wybranyObiekt.name +
                      " | pozycja: " + hit.point);
        }
        else {
            PrzywrocKolor();   // kliknięcie w pustą przestrzeń
        }
    }
```

```
    void PodswietlObiekt() {
        var renderer = wybranyObiekt.GetComponent<Renderer>();
        if (renderer == null) return;
        oryginalnyKolor = renderer.material.color;
        renderer.material.color = kolorWyboru;
    }

    void PrzywrocKolor() {
        if (wybranyObiekt == null) return;
        var renderer = wybranyObiekt.GetComponent<Renderer>();
        if (renderer != null)
            renderer.material.color = oryginalnyKolor;
        wybranyObiekt = null;
    }
}
```

4. Upewniamy się, że wszystkie obiekty klikalne posiadają komponent Collider (np. Box Collider, Sphere Collider).
5. Uruchamiamy scenę i klikamy na kolejne obiekty. Obs erwujemy zmianę koloru.
6. Sprawdzamy w konsoli (Window → General → Console) wypisywane komunikaty Debug.Log.

Krok 2: Rozszerzenie: interakcja z wybranymi obiektami

Dodajemy do trzech obiektów w scenie skrypt reagujący na wybór. Tworzymy skrypt InteraktywnyObiekt:

```
using UnityEngine;
public class InteraktywnyObiekt : MonoBehaviour {
    [SerializeField] private string nazwaCzynnosci = "Aktywuj";
    // Metoda publiczna: wywoływana z zewnątrz przez WyborObiektu
    public void Aktywuj() {
        Debug.Log(gameObject.name + ": " + nazwaCzynnosci);
        // Tu w przyszłości dodamy właściwą logikę interakcji
    }
}
```

Następnie w skrypcie WyborObiektu, po linii Debug.Log, dodajemy wywołanie publicznej metody znalezionego obiektu:

```
// Dodajemy wewnątrz bloku if (Physics.Raycast ...), po Debug.Log:
var interakcja = wybranyObiekt.GetComponent<InteraktywnyObiekt>();
if (interakcja != null)
    interakcja.Aktywuj();
```

4.3. Budowa interfejsu użytkownika: Canvas i TextMeshPro

Przystępujemy do budowy warstwy wizualnej interfejsu. Stworzymy HUD (Heads-Up Display) zawierający wyświetlany licznik kliknięć oraz etykietę z nazwą aktualnie wybranego obiektu.

Krok 1: Tworzenie Canvas

7. W Hierarchy: prawym przyciskiem → UI → Canvas. Unity automatycznie doda również obiekt EventSystem (niezbędny do obsługi kliknięć w UI).
8. Zaznaczamy Canvas → Inspector → Canvas komponent: Render Mode ustaw na Screen Space – Overlay.
9. Zaznaczamy Canvas Scaler: UI Scale Mode → Scale With Screen Size, Reference Resolution: 1920 × 1080. Zapewnia to skalowanie UI na różnych rozdzielczościach.

Krok 2: Panel tła

10. Prawym przyciskiem na Canvas → UI → Panel. Zmień nazwę na PanelHUD.

11. W Rect Transform: Anchor Presets → kliknij ikonę w lewym górnym rogu → wybierz top-left (zakotwiczenie).
12. Ustaw Width: 340, Height: 130, Pos X: 20, Pos Y: -20.
13. W Image → Color ustaw kolor ciemnoszary z przezroczystością (Alpha ok. 160), aby panel nie przysłaniał sceny.

Krok 3: Elementy tekstowe TextMeshPro

14. Prawym przyciskiem na PanelHUD → UI → Text – TextMeshPro. Zmień nazwę na TekstLicznik.
15. W TMP – Text: wpisz Kliknięcia: 0 (to jest tekst startowy).
16. Font Size: 22, kolor biały, Alignment: Left.
17. Rect Transform: ustaw pozycję tak, by tekst zmieścił się w panelu (np. Pos Y: -25, Width: 300, Height: 40).
18. Powtórz dla drugiego elementu: TekstWybrany: z tekstem startowym Wybrany: brak.

Krok 4: Przycisk

19. Prawym przyciskiem na Canvas (poza PanelHUD) → UI → Button – TextMeshPro. Zmień nazwę na PrzyciskReset.
20. Zakotwiczenie: bottom-center. Pos Y: 40, Width: 200, Height: 50.
21. Rozłóż przycisk w hierarchii → zaznacz obiekt Text (TMP) wewnątrz → zmień tekst na Resetuj licznik.

4.4. Połączenie interfejsu z logiką aplikacji

Elementy UI są teraz gotowe wizualnie, ale nieruchome: licznik nie zmienia się, przycisk nic nie robi. Teraz napiszemy skrypt zarządzający HUDem i połączymy go z wcześniejszą logiką.

Krok 1: Skrypt ZarzadcaHUD

Tworzymy nowy skrypt C# o nazwie ZarzadcaHUD i przyłączamy go do obiektu GameManager:

```
using UnityEngine;
using TMPro; // przestrzeń nazw TextMeshPro

public class ZarzadcaHUD : MonoBehaviour
{
    // Referencje do elementów UI: przeciągamy je w Inspectorze
    [SerializeField] private TMP_Text tekstLicznik;
    [SerializeField] private TMP_Text tekstWybrany;

    private int licznikKlikniec = 0;

    // Metoda wywoływana przez WyborObiektu przy każdym kliknięciu
    public void AktualizujWybor(string nazwaObiektu)
    {
        licznikKlikniec++;
        tekstLicznik.text = "Kliknięcia: " + licznikKlikniec;
        tekstWybrany.text = "Wybrany: "    + nazwaObiektu;
    }

    // Metoda wywoływana przez przycisk Resetuj
    public void Resetuj()
    {
        licznikKlikniec     = 0;
        tekstLicznik.text   = "Kliknięcia: 0";
        tekstWybrany.text   = "Wybrany: brak";
    }
}
```

Krok 2: Przyłączenie referencji w Inspectorze

22. Zaznaczamy GameManager w hierarchii.
23. W Inspektorze w komponencie ZarzadcaHUD widzimy pola Tekst Licznik i Tekst Wybrany.
24. Przeciągamy obiekt TekstLicznik z hierarchii do pola Tekst Licznik, analogicznie dla TekstWybrany.

Krok 3: Aktualizacja skryptu WyborObiektu

Musimy połączyć logikę klikania z HUDem. Dodajemy do skryptu WyborObiektu referencję do ZarzadcaHUD:

```
// Dodajemy pole na początku klasy WyborObiektu:
[SerializeField] private ZarzadcaHUD hud;

// Wewnątrz bloku if (Physics.Raycast ...), po wywołaniu Aktywuj():
if (hud != null)
    hud.AktualizujWybor(wybranyObiekt.name);
```

25. W Inspektorze komponentu WyborObiektu przeciagamy GameManager do pola Hud.

Krok 4: Podłączenie przycisku

26. Zaznaczamy obiekt PrzyciskReset w hierarchii.
27. W Inspektorze w komponencie Button → sekcja On Click () → klikamy +.
28. Przeciągamy GameManager do pola obiektu (None (Object)).
29. Z listy funkcji wybieramy ZarzadcaHUD → Resetuj().
30. Uruchamiamy scenę, klikamy na obiekty i obserwujemy aktualizację HUDu oraz działanie przycisku Reset.

> **Wzorzec UnityEvent**
> Przycisk komunikuje się ze skryptem przez system zdarzeń Unity (UnityEvent), a nie przez bezpośrednią referencję do skryptu. To celowy project: przycisk nie musi wiedzieć nic o ZarzadcaHUD: jedynie wysyła sygnał 'kliknięto mnie', a kto go obsłuży, definiujemy w Inspektorze. To przykład luźnego powiązania (loose coupling).

4.5. Feedback wizualny i dźwiękowy

Dobra aplikacja interaktywna zawsze potwierdza działania użytkownika. Zaimplementujemy dwa rodzaje feedbacku: wizualny (animacja przycisku) i dźwiękowy (krótki sygnał po kliknięciu).

Krok 1: Efekt hover na przycisku

31. Zaznaczamy PrzyciskReset.
32. W komponencie Button → Transition wybieramy Color Tint (domyślne).
33. Konfigurujemy kolory:
    - Normal Color: ciemnoszary (kolor spoczynkowy),
    - Highlighted Color: jasnofioletowy (kolor po najechaniu myszą),
    - Pressed Color: ciemniejszy fiolet (kolor po kliknięciu),
    - Disabled Color: szary (kolor zablokowanego przycisku).
34. Testujemy najechi wanie i klikanie przycisku w trybie Play.

Krok 2: Sygnał dźwiękowy

35. Pobieramy krótki efekt dźwiękowy (kliknięcie, pik) z serwisu freesound.org lub z zestawu zasobów Kenney (kenney.nl/assets/interface-sounds). Plik musi być w formacie .wav lub .ogg.
36. Kopiujemy plik dźwiękowy do folderu Assets/Sounds (tworzymy ten folder).
37. Do obiektu GameManager dodajemy komponent Audio Source (Add Component → Audio → Audio Source). Odznaczamy Play On Awake.
38. Rozszerzamy skrypt ZarzadcaHUD:

```
// Dodajemy pole na początku klasy:
[SerializeField] private AudioSource audioSource;
[SerializeField] private AudioClip dzwiekKlikniecia;

// Dodajemy na końcu metody AktualizujWybor():
if (audioSource != null && dzwiekKlikniecia != null)
    audioSource.PlayOneShot(dzwiekKlikniecia);
```

39. W Inspektorze przypisujemy Audio Source i plik dźwiękowy do odpowiednich pól.
40. Testujemy: każde kliknięcie na obiekt powinno odtworzyć krótki sygnał.

5. Analiza ergonomii: ocena własnego interfejsu

Zbudowana przez nas aplikacja jest gotowa technicznie. Pora ocenić ją z perspektywy użytkownika, który nie zna jej kodu.

5.1. Szablon analizy UX

W pliku dokumentacji (folder Documentation) uzupełniamy krótką analizę według następującego szablonu:

Kryterium UX	Pytanie kontrolne	Ocena i uzasadnienie
Widoczność	Czy użytkownik wie, które obiekty są interaktywne?	(wpisz ocenę i uzasadnienie)
Feedback	Czy każda akcja ma natychmiastową odpowiedź?	
Mapowanie	Czy sterowanie jest zgodne z intuicją?	
Czytelność UI	Czy tekst jest wystarczająco duży i kontrastowy?	
Spójność	Czy kolory i styl są jednolite w całej aplikacji?	
Obsługa błędów	Co się dzieje, gdy użytkownik kliknie w pustą przestrzeń?	

Przygotowujemy również krótką listę (min. 3 pozycje) potencjalnych ulepszeń UX, które należałoby wprowadzić, gdyby aplikacja była rozwijana dalej. Nie wymagamy implementacji tych ulepszeń: wystarczy ich opisanie z uzasadnieniem.

6. Lista kontrolna wykonanych zadań

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

☐	Scena zawiera płaską podłogę i co najmniej trzy obiekty 3D z komponentami Collider.
☐	Skrypt SterowanieObiektem lub SterowanieKamera działa poprawnie (ruch WASD, obrót).
☐	Skrypt WyborObiektu poprawnie rzutuje promień i podświetla kliknięty obiekt.

☐	Co najmniej trzy obiekty mają komponent InteraktywnyObiekt z wywołaniem metody Aktywuj().
☐	Canvas skonfigurowany w trybie Screen Space – Overlay, skalowany do 1920×1080.
☐	HUD zawiera TekstLicznik i TekstWybrany aktualizowane przy każdym kliknięciu.
☐	Przycisk Resetuj działa i zeruje licznik przez system UnityEvent.
☐	Feedback wizualny: zmiana koloru klikniętego obiektu działa poprawnie.
☐	Feedback dźwiękowy: sygnał dźwiękowy odtwarzany przy każdym kliknięciu.
☐	Analiza UX uzupełniona w dokumentacji (tabela + lista potencjalnych ulepszeń).

7. Materiały uzupełniające

Zasób	Opis
Unity Manual: Input Manager	Dokumentacja systemu wejścia, opis osi i przycisków.
Unity Manual: UI Toolkit / uGUI	Dokumentacja systemu UI: Canvas, Button, TextMeshPro.
Unity Manual: Physics.Raycast	Pełna dokumentacja metody raycastingu z przykładami.
The Design of Everyday Things (D. Norman)	Książka z listy literatury kursu: rozdziały 1–2 o mapowaniu i feedbacku.
Game Programming Patterns: rozdział Event Queue	Wzorzec kolejki zdarzeń: rozszerzenie UnityEvent na własne systemy.
freesound.org	Baza darmowych efektów dźwiękowych na licencjach CC.