POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ BUDOWY MASZYN I LOTNICTWA
POLITECHNIKI RZESZOWSKIEJ

Techniki wirtualnej rzeczywistości w mechatronice

Laboratorium nr 4-5

Temat:

Podstawy obsługi myszki, animacji i dźwięku w Unity

Katedra Mechaniki Stosowanej i Robotyki

Wydział Budowy Maszyn i Lotnictwa, Politechnika Rzeszowska

27 marca 2026

Opracował: dr inż. Jakub Wiech

Liczba zajęć przeznaczona na temat: 2

1. Wstęp

Celem ćwiczenia jest zapoznanie się z podstawami interakcji w Unity: obsługą wejścia z myszki, przygotowaniem prostych animacji oraz odtwarzaniem dźwięku. W trakcie zajęć student utworzy scenę demonstracyjną, w której obiekty reagują na kliknięcie i najechanie kursorem, poruszają się zgodnie z animacją oraz emitują dźwięki.

2. Plan realizacji ćwiczenia

Uwaga organizacyjna: Zajęcia mają charakter praktyczny. Kod i konfiguracje będą przygotowywane krok po kroku zgodnie z instrukcją prowadzącego.

Przebieg ćwiczenia (instruktaż prowadzącego):

1. Przygotowanie projektu i sceny roboczej w Unity.
2. Konfiguracja obiektów do testowania interakcji myszką.
3. Dodanie komponentów odpowiedzialnych za animację obiektów.
4. Dodanie i konfiguracja źródeł dźwięku.
5. Integracja skryptów: kliknięcie myszy, animacja i dźwięk.
6. Testowanie działania w `Play mode` i analiza efektów.
7. Podsumowanie i omówienie najczęstszych błędów.

3. Programy z ćwiczenia

Programy można pobrać korzystając z aplikacji Adobe Acrobat.

3..1 Program 1 — `CameraOrbit.cs`

Skrypt realizuje sterowanie kamerą myszką: obrót wokół obiektu przy wciśniętym prawym przycisku oraz przybliżanie/oddalanie rolką. Dodatkowo ogranicza kąt pionowy i dystans kamery.

Pobierz plik programu: Pobierz CameraOrbit.cs

3..2 Program 2 — `ConfiguratorManager.cs`

Skrypt zarządza interakcją użytkownika z obiektami w scenie. Wykrywa obiekt pod kursorem, zaznacza element po kliknięciu lewym przyciskiem myszy oraz uruchamia dla niego animację i dźwięk.

Pobierz plik programu: Pobierz ConfiguratorManager.cs

3..3 Program 3 — `ConfigurablePart.cs`

Skrypt opisuje pojedynczy obiekt konfigurowalny. Odpowiada za stan zaznaczenia i podświetlenia, wyzwalanie animacji przez `Animator` oraz odtwarzanie dźwięku kliknięcia przez `AudioSource`.

Pobierz plik programu: Pobierz ConfigurablePart.cs

4. Cel i zadania do wykonania

Utwórz scenę demonstracyjną i zaimplementuj trzy elementy funkcjonalne: reakcję obiektu na myszkę, prostą animację uruchamianą zdarzeniem oraz odtwarzanie dźwięku zsynchronizowane z interakcją. Po wykonaniu całego ćwiczenia powstaną trzy skrypty: `CameraOrbit.cs`, `ConfigurablePart.cs`, `ConfiguratorManager.cs`.

Końcowy system będzie działał tak:

- kamera obraca się wokół obiektu,
- kamera przybliża i oddala widok,
- wybrane części obiektu reagują na kliknięcie,
- część może być podświetlana po najechaniu kursorem,
- kliknięcie zaznacza część,
- kliknięcie może uruchomić animację i odtworzyć dźwięk,
- kliknięcie poza obiektem usuwa zaznaczenie.

4..1 ETAP 0 — przygotowanie obiektów na scenie

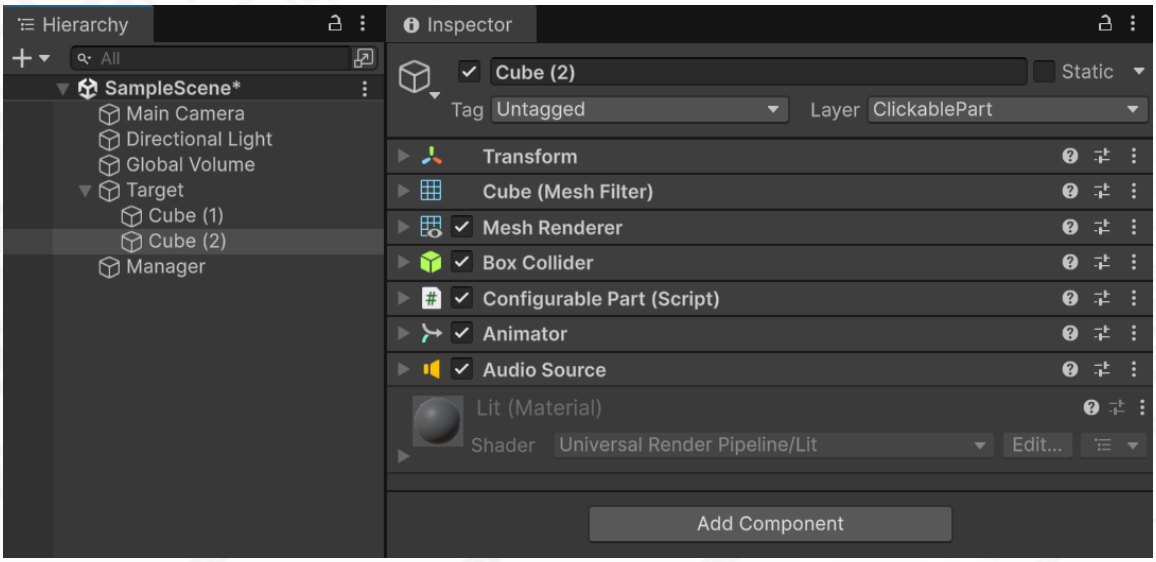
Utworzyć nowy projekt Unity z szablonu Universal 3D. Scena powinna zawierać Kamerę i trzy obiekty -> 1 pusty, do którego są przypisane dwa z modelem 3D np. Cube. Skrypty `CameraOrbit.cs` i `ConfiguratorManager.cs` przypisujemy do Kamery a skrypt `ConfigurablePart.cs` do obiektu, który będzie klikany. W skryptach w inspektorze należy przypisać obiekty takie jakie skrypt potrzebuje. Jako warstwę Layer należy wybrać warstwę Default.

Najpierw należy upewnić się czy działa obsługa kamery z użyciem skryptu `CameraOrbit.cs`. Następnie, korzystając ze skryptów `ConfigurablePart.cs`, `ConfiguratorManager.cs` sprawdzić czy podświetla się obiekt po najechaniu na niego wskaźnikiem myszy. Jeżeli obiekt podświetla się to należy sprawdzić czy po kliknięciu na niego wyświetla się jego nazwa w terminalu Unity. Kolejno należy utworzyć animację dla obiektu oraz przypisać mu dźwięk.

Przygotowanie obiektu na scenie

Na początku należy upewnić się, że obiekt, który ma reagować na kliknięcie, jest poprawnie przygotowany. Obiekt powinien mieć co najmniej: model 3D lub jego część widoczną na scenie, `Collider`, aby można było kliknąć w obiekt, skrypt `ConfigurablePart`, `Animator`, `AudioSource`.

Jeżeli obiekt składa się z kilku części, to każda część, która ma reagować na kliknięcie, powinna mieć własny collider i skrypt `ConfigurablePart` albo być dzieckiem obiektu, który taki skrypt posiada.



Rys. 1: Widok obiektu w `Hierarchy` oraz `Inspector` z dodanym `Collider` i `ConfigurablePart`.

4..2 ETAP 1 — budowa skryptu `CameraOrbit.cs`

Najpierw tworzony jest wyłącznie skrypt kamery.

Krok 1. Utwórz plik `CameraOrbit.cs`

Tę wstawkę wklej jako całą zawartość nowego pliku `CameraOrbit.cs`.

Listing 1: Początek pliku CameraOrbit.cs

```
// Podstawowe przestrzenie nazw Unity i systemu wejścia
using UnityEngine;
using UnityEngine.InputSystem;
```

```
// Skrypt odpowiadający za orbitowanie kamery wokół celu
public class CameraOrbit : MonoBehaviour
{
}
```

To jest szkielet skryptu kamery. Dyrektywa `using UnityEngine` udostępnia podstawowe klasy Unity, a `using UnityEngine.InputSystem` pozwala używać nowego systemu wejścia. Ten etap przygotowuje miejsce na logikę obrotu i przybliżania kamery. pola konfiguracyjne, metody obsługi wejścia myszy oraz metodę aktualizacji pozycji kamery względem celu.

Krok 2. Dodaj pola opisujące kamerę

Tę wstawkę umieść w pliku `CameraOrbit.cs`, wewnątrz klasy `CameraOrbit`, bezpośrednio pod deklaracją klasy.

Listing 2: Pola kamery

```
// Obiekt, wokół którego obraca się kamera
[Header("Obiekt, wokół którego obraca się kamera")]
[SerializeField] private Transform target;

// Parametry odległości kamery
[Header("Odległość kamery od obiektu")]
[SerializeField] private float distance = 5f;
[SerializeField] private float minDistance = 2f;
[SerializeField] private float maxDistance = 10f;

// Czułość obrotu i zoomu
[Header("Szybkość obrotu i zoomu")]
[SerializeField] private float rotationSpeed = 0.2f;
[SerializeField] private float zoomSpeed = 0.01f;

// Limity kąta pionowego
[Header("Ograniczenie obrotu w pionie")]
[SerializeField] private float minPitch = -20f;
[SerializeField] private float maxPitch = 80f;

// Aktualne kąty kamery
private float yaw = 0f;
private float pitch = 20f;
```

Wstawka definiuje wszystkie parametry potrzebne do pracy kamery orbitalnej: cel, dystans, limity zoomu, czułość sterowania i limity kąta pionowego. Dzięki polom `SerializeField` parametry można regulować z poziomu `Inspectora` bez zmiany kodu. wartości `yaw` i `pitch` będą modyfikowane ruchem myszy i wykorzystane do wyliczania nowej pozycji kamery.

Krok 3. Dodaj metodę `Update()`

Tę metodę wklej w pliku `CameraOrbit.cs`, wewnątrz klasy `CameraOrbit`, poniżej pól kamery.

```
private void Update()
{
    // Obsługa obrotu
    RotateCamera();
    // Obsługa przybliżania i oddalania
    ZoomCamera();
    // Ustawienie pozycji kamery
    UpdateCameraPosition();
}
```

Metoda `Update()` jest wykonywana co klatkę i wywołuje trzy kroki logiki kamery. Rozdziela działanie na czytelne etapy: obrót, zoom i przeliczenie położenia. docelowo każda z wywoływanych metod dostanie własną odpowiedzialność i łatwiejsze debugowanie.

Krok 4. Dodaj metodę `RotateCamera()`

Tę metodę dodaj w pliku `CameraOrbit.cs`, wewnątrz klasy `CameraOrbit`, jako osobną metodę pomocniczą.

```
private void RotateCamera()
{
    // Jeśli mysz jest niedostępna, pomijamy sterowanie
    if (Mouse.current == null)
        return;

    // Obrót tylko przy wciśniętym prawym przycisku
    if (Mouse.current.rightButton.isPressed)
    {
        // Różnica ruchu myszy od poprzedniej klatki
        Vector2 mouseDelta = Mouse.current.delta.ReadValue();
        // Obrót poziomy
        yaw = yaw + mouseDelta.x * rotationSpeed;
        // Obrót pionowy
        pitch = pitch - mouseDelta.y * rotationSpeed;
        // Ograniczenie obrotu pionowego
        pitch = Mathf.Clamp(pitch, minPitch, maxPitch);
    }
}
```

Metoda odczytuje ruch myszy tylko wtedy, gdy prawy przycisk jest wciśnięty, a następnie aktualizuje kąty `yaw` i `pitch`. Zapewnia intuicyjne obracanie widoku wokół obiektu oraz ogranicza pionowy obrót funkcją `Mathf.Clamp`. zaktualizowane kąty zostaną użyte przy obliczaniu wektora kierunku i pozycji kamery.

Krok 5. Dodaj metodę `ZoomCamera()`

Tę metodę dodaj w pliku `CameraOrbit.cs`, wewnątrz klasy `CameraOrbit`, obok `RotateCamera()`.

```
private void ZoomCamera()
```

```
{
    // Jeśli mysz jest niedostępna, pomijamy zoom
    if (Mouse.current == null)
        return;

    // Odczyt ruchu rolki myszy
    Vector2 scroll = Mouse.current.scroll.ReadValue();
    // Zmiana odległości kamery od obiektu
    distance = distance - scroll.y * zoomSpeed;
    // Ograniczenie odległości do zadanego zakresu
    distance = Mathf.Clamp(distance, minDistance, maxDistance);
}
```

Metoda pobiera sygnał z rolki myszy i na jego podstawie zmienia odległość kamery od celu. Umożliwia płynne przybliżanie i oddalanie oraz chroni przed zbyt małym lub zbyt dużym dystansem. nowa wartość `distance` zostanie uwzględniona przy ustawianiu pozycji kamery w przestrzeni.

Krok 6. Dodaj metodę `UpdateCameraPosition()`

Tę metodę dodaj w pliku `CameraOrbit.cs`, wewnątrz klasy `CameraOrbit`, pod metodami obsługi wejścia.

```
private void UpdateCameraPosition()
{
    // Przeliczenie kątów na obrót
    Quaternion rotation = Quaternion.Euler(pitch, yaw, 0f);
    // Kierunek patrzenia kamery
    Vector3 direction = rotation * Vector3.forward;
    // Wyliczenie pozycji kamery względem celu
    Vector3 cameraPosition = target.position - direction * distance;

    // Ustawienie nowej pozycji i skierowanie na cel
    transform.position = cameraPosition;
    transform.LookAt(target.position);
}
```

Metoda zamienia kąty obrotu na kwaternion, wylicza kierunek patrzenia i przesuwa kamerę na okręgu wokół celu. To kluczowy etap, który faktycznie ustawia pozycję kamery i orientuje ją na obiekt `target`. finalnie skrypt `CameraOrbit.cs` będzie kompletną implementacją sterowania kamerą 3D.

Krok 7. Co sprawdzić

Dodaj `CameraOrbit` do `Main Camera`, przypisz `Target`, uruchom scenę i sprawdź obrót prawym przyciskiem oraz zoom rolką.

4..3 ETAP 2 — równoległa budowa `ConfigurablePart.cs` i `ConfiguratorManager.cs`

Od tego momentu rozwijane są dwa pliki równocześnie.

Krok 1. Utwórz dwa pliki

Utwórz `ConfigurablePart.cs` i `ConfiguratorManager.cs`.

ETAP 2A — identyfikacja części i kliknięcie

Tę wstawkę wklej jako zawartość nowego pliku `ConfigurablePart.cs`.

Listing 3: ConfigurablePart.cs — wersja początkowa

```
using UnityEngine;

// Skrypt pojedynczej części konfigurowalnej
public class ConfigurablePart : MonoBehaviour
{
    // Nazwa części widoczna w UI lub logach
    [Header("Podstawowe informacje o części")]
    [SerializeField] private string partName = "Nowa część";

    // Odczyt nazwy części
    public string PartName
    {
        get { return partName; }
    }
}
```

To początkowa wersja skryptu przypinanego do pojedynczej części konfigurowalnej. Udostępnia nazwę części `PartName`, którą można pokazać w UI lub logach diagnostycznych. pola stanu zaznaczenia i najechania, a także metody animacji i odtwarzania dźwięku.

Tę wstawkę wklej jako zawartość nowego pliku `ConfiguratorManager.cs`.

Listing 4: ConfiguratorManager.cs — wersja początkowa

```
using UnityEngine;
using UnityEngine.InputSystem;

// Menedżer interakcji użytkownika z obiektami
public class ConfiguratorManager : MonoBehaviour
{
    // Kamera używana do rzutowania promienia
    [SerializeField] private Camera mainCamera;
    // Maksymalny zasięg raycastu
    [SerializeField] private float maxRayDistance = 100f;
    // Warstwa klikalnych obiektów
    [SerializeField] private LayerMask clickableLayer;
}
```

To szkielet menedżera interakcji dla całej sceny. Trzyma referencję do kamery oraz parametry raycastu wykorzystywane do wykrywania klikanych obiektów. logikę hover, kliknięcia, zaznaczania elementu i wywołań animacji oraz dźwięku.

Krok 2. Dodaj zaznaczanie po kliknięciu

Tę wstawkę dopisz w pliku `ConfigurablePart.cs`, wewnątrz klasy `ConfigurablePart`, pod istniejącymi polami.

Listing 5: Wstawka: Select/Deselect i UpdateColor

```
// Renderer obiektu do zmiany wyglądu
private Renderer objectRenderer;
// Flaga informująca, czy obiekt jest zaznaczony
private bool isSelected = false;

public void Select()
{
    // Włączenie zaznaczenia
    isSelected = true;
    // Odświeżenie koloru
    UpdateColor();
}

public void Deselect()
{
    // Wyłączenie zaznaczenia
    isSelected = false;
    // Odświeżenie koloru
    UpdateColor();
}
```

Dodane są pola renderera i stanu zaznaczenia oraz metody `Select()` i `Deselect()`. Te metody centralizują zmianę stanu i wymuszają odświeżenie koloru przez `UpdateColor()`. pełna logika koloru uwzględniająca jednocześnie hover i select.

Tę wstawkę dopisz w pliku `ConfiguratorManager.cs`, wewnątrz klasy `ConfiguratorManager`, pod polami konfiguracyjnymi.

Listing 6: Wstawka: CheckClick w ConfiguratorManager

```
// Aktualnie zaznaczona część
private ConfigurablePart selectedPart;

private void CheckClick()
{
    // Brak myszy - brak obsługi kliknięcia
    if (Mouse.current == null) return;
    // Reakcja tylko na pojedyncze kliknięcie LPM
    if (Mouse.current.leftButton.wasPressedThisFrame == false) return;

    // Pozycja kursora na ekranie
    Vector2 mousePosition = Mouse.current.position.ReadValue();
```

```
    // Ray z kamery przez pozycję kursora
    Ray ray = mainCamera.ScreenPointToRay(mousePosition);

    // Sprawdzenie trafienia w klikalny obiekt
    if (Physics.Raycast(ray, out RaycastHit hit, maxRayDistance, clickableLayer))
    {
        // Pobranie części z trafionego obiektu lub rodzica
        ConfigurablePart clickedPart = hit.collider.GetComponentInParent<ConfigurablePart
    >();
        if (clickedPart != null)
        {
            // Odznaczenie poprzednio wybranego elementu
            if (selectedPart != null) selectedPart.Deselect();
            // Zapis i zaznaczenie nowo kliknięte części
            selectedPart = clickedPart;
            selectedPart.Select();
        }
    }
}
```

Menedżer nasłuchuje lewego kliknięcia, tworzy promień z pozycji kursora i sprawdza trafienie w warstwie klikalnej. Pozwala wybrać jedną część naraz oraz przełączyć zaznaczenie z poprzedniego obiektu na nowy. po zaznaczeniu obiektu metoda zostanie rozszerzona o uruchamianie animacji i dźwięku.

ETAP 2B — hover

Tę wstawkę dopisz w pliku `ConfigurablePart.cs`, wewnątrz klasy `ConfigurablePart`, obok pól i metod zaznaczenia.

Listing 7: Wstawka: hover w ConfigurablePart

```
// Flaga informująca, czy kursor jest nad obiektem
private bool isHovered = false;

public void SetHover(bool value)
{
    // Ustawienie stanu hover
    isHovered = value;
    // Odświeżenie koloru
    UpdateColor();
}
```

Dodaje stan `isHovered` i metodę `SetHover(bool value)`. Pozwala menedżerowi sterować podświetleniem obiektu przy najechaniu kursorem. metoda `UpdateColor()` będzie rozróżniała co najmniej trzy stany: normalny, hover i selected.

Tę wstawkę dopisz w pliku `ConfiguratorManager.cs`, wewnątrz klasy `ConfiguratorManager`, tak aby `Update()` wywoływało `CheckHover()` i `CheckClick()`.

Listing 8: Wstawka: CheckHover w ConfiguratorManager

```
// Aktualnie podświetlona część
private ConfigurablePart hoveredPart;

private void Update()
{
    // Sprawdzenie najechania kursorem
    CheckHover();
    // Sprawdzenie kliknięcia
    CheckClick();
}
```

Wstawka dodaje śledzenie aktualnie najechanej części oraz wywołanie kontroli hover w każdej klatce. Zapewnia bieżącą reakcję wizualną na ruch kursora niezależnie od kliknięcia. właściwa implementacja `CheckHover()` będzie ustawiała i resetowała podświetlenie obiektów. Ta wstawka jest zgodna z wersją końcową programu 2 (`ConfiguratorManager.cs`).

ETAP 2C — usuwanie zaznaczenia po kliknięciu poza obiektem

Tę metodę dopisz w pliku `ConfiguratorManager.cs`, wewnątrz klasy `ConfiguratorManager`, jako metodę pomocniczą do resetu stanu.

Listing 9: Wstawka: ClearSelection

```
private void ClearSelection()
{
    // Jeśli coś było zaznaczone, odznacz to
    if (selectedPart != null)
    {
        // Zmiana stanu wizualnego
        selectedPart.Deselect();
        // Wyczyszczenie referencji
        selectedPart = null;
    }
}
```

Metoda usuwa zaznaczenie bieżącej części i czyści referencję `selectedPart`. Utrzymuje spójny stan systemu po kliknięciu w puste miejsce sceny. zostanie wywołana w logice kliknięcia, gdy raycast nie trafi w poprawny obiekt.

ETAP 2D — dodanie animacji

Tę wstawkę dopisz w pliku `ConfigurablePart.cs`, wewnątrz klasy `ConfigurablePart`, pod wcześniejszymi polami i metodami.

Listing 10: Wstawka: pola i metoda animacji w ConfigurablePart

```
// Ustawienia animacji przypisane do części
[Header("Animacja")]
[SerializeField] private Animator animator;
[SerializeField] private string animationTriggerName = "PlayAnimation";
```

```
public void PlayClickAnimation()
{
    // Brak komponentu Animator - brak animacji
    if (animator == null)
        return;

    // Uruchomienie triggera animacji
    animator.SetTrigger(animationTriggerName);
}
```

Dodaje pola animacji i metodę ustawiającą trigger w komponencie `Animator`. Umożliwia uruchamianie animacji przypisanej do konkretnej części po interakcji użytkownika. ostatecznie każda część może mieć własny animator i własną nazwę triggera.

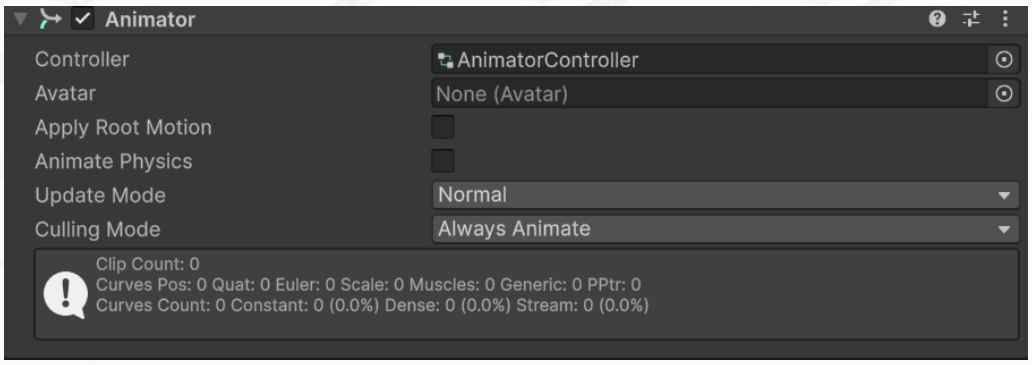
Dodanie komponentu Animator

Aby obiekt mógł odtwarzać animację, trzeba dodać do niego komponent `Animator`.

Krok po kroku:

1. Zaznacz obiekt w oknie `Hierarchy`.
2. Przejdź do okna `Inspector`.
3. Kliknij przycisk `Add Component`.
4. Wpisz `Animator`.
5. Wybierz komponent `Animator` z listy.

Po dodaniu komponentu w `Inspectorze` pojawi się pole `Controller`. Na razie może ono być puste.



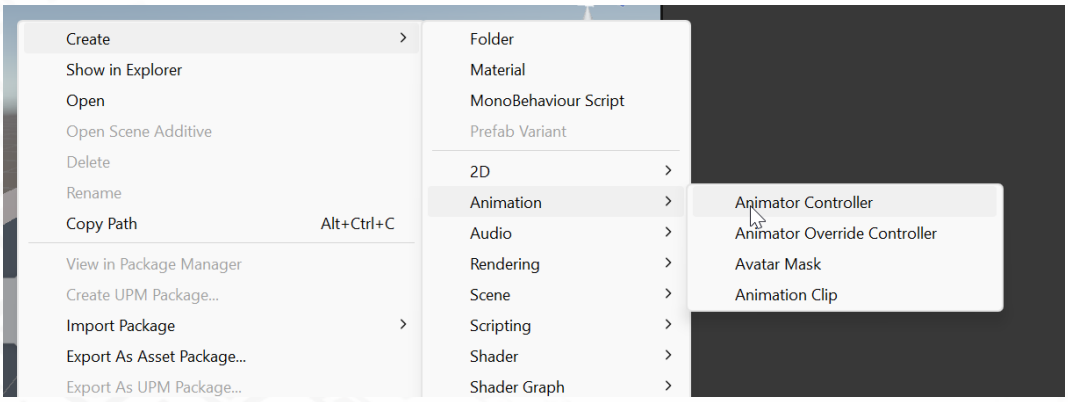
Rys. 2: Dodawanie komponentu `Animator` w `Inspectorze`.

Utworzenie kontrolera animacji

`Animator` potrzebuje kontrolera animacji, czyli pliku typu `Animator Controller`, który opisuje stany animacji i przejścia między nimi.

Krok po kroku:

1. W oknie `Project` kliknij prawym przyciskiem myszy.
2. Wybierz: `Create -> Animator Controller`.
3. Nadaj plikowi nazwę, na przykład: `PartAnimatorController`.

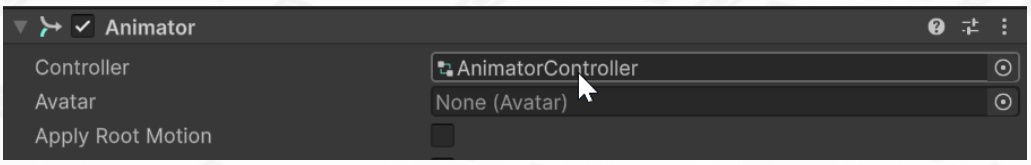


Rys. 3: Utworzenie `Animator Controller` w oknie `Project`.

Następnie przypisz ten kontroler do pola `Controller` w komponencie `Animator`.

Przypisanie kontrolera:

1. Zaznacz obiekt z komponentem `Animator`.
2. W polu `Controller` przeciągnij utworzony wcześniej `Animator Controller`.



Rys. 4: Przypisanie `Animator Controller` do pola `Controller`.

Utworzenie klipu animacji

W kolejnym kroku należy przygotować samą animację. Najprostsza wersja polega na zapisaniu zmian położenia, obrotu albo skali obiektu w czasie.

Do ćwiczenia najlepiej przygotować prostą animację, na przykład: lekkie otwarcie elementu, obrót części, przesunięcie fragmentu obiektu, uniesienie klapki lub pokrywy.

Krok po kroku:

1. Zaznacz obiekt, który ma się animować.

2. Otwórz okno: `Window -> Animation -> Animation`.
3. Jeżeli obiekt nie ma jeszcze klipu animacji, Unity zaproponuje jego utworzenie.
4. Kliknij `Create`.
5. Zapisz plik animacji, na przykład pod nazwą: `PartClickAnimation`.

Po utworzeniu klipu pojawi się okno osi czasu animacji.

Rys. 5: Okno `Animation` i tworzenie nowego klipu.

Nagranie prostej animacji

Po utworzeniu klipu można nagrać prostą animację.

Przykład: obrót elementu

1. W oknie `Animation` włącz tryb nagrywania, klikając czerwony przycisk `Record`.
2. Ustaw wskaźnik czasu na początek animacji.
3. Pozostaw obiekt w pozycji początkowej.
4. Przesuń wskaźnik czasu dalej, na przykład na 1 sekundę.
5. Zmień obrót obiektu w `Inspectorze` lub bezpośrednio na scenie.
6. Unity automatycznie zapisze klatki kluczowe.
7. Wyłącz tryb nagrywania.

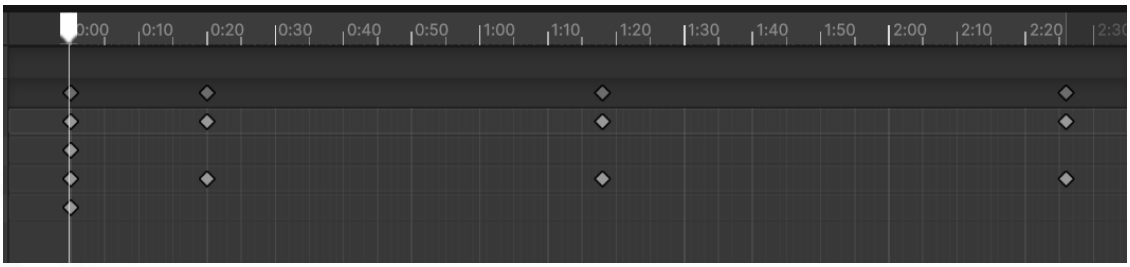
W ten sposób powstaje najprostsza animacja.

Wskazówka dydaktyczna: na potrzeby ćwiczenia warto stworzyć animację krótką i wyraźną, aby student od razu widział efekt kliknięcia. Najlepiej, aby trwała od około 0.5 s do 1.5 s.

Sprawdzenie Animator Controller

Po utworzeniu klipu animacji Unity zwykle automatycznie doda go do `Animator Controller` jako stan.

Aby to sprawdzić:

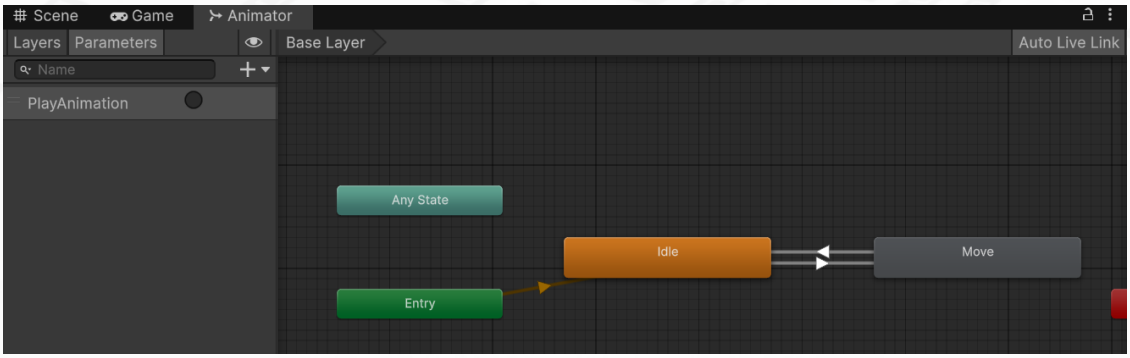


Rys. 6: Nagrywanie animacji i dodawanie klatek kluczowych.

1. Otwórz okno: `Window -> Animation -> Animator`.
2. Zobacz, czy w oknie znajduje się stan z nazwą utworzonej animacji.
3. Jeżeli stan istnieje, oznacza to, że kontroler widzi już klip animacji.

W ćwiczeniu należy dodać stan spoczynkowy i stan reakcji na kliknięcie. Aby utworzyć stan spoczynkowy `Idle` oraz stan reakcji na kliknięcie `Move` należy:

1. Usunąć domyślny stan animacji (O ILE ISTNIEJE) z okna Animatora klikając na niego prawym przyciskiem myszy (PPM) i klikając opcję Delete.
2. Utworzyć nowe stany: `PPM -> Create State -> Empty`.
3. W inspektorze nowego stanu należy upewnić się czy jest przypisana animacja dla stanu `Move` a dla stanu `Idle` brak animacji (parametr Motion).
4. Następnie należy utworzyć między stanami ścieżki Tranzycji (poniżej instrukcja)



Rys. 7: Okno `Animator` ze stanem animacji.

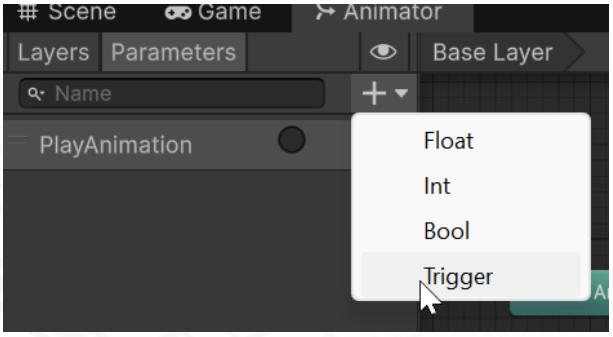
Dodanie parametru wyzwalającego animację

Aby uruchamiać animację ze skryptu, najwygodniej dodać parametr typu `Trigger`.

Krok po kroku:

1. Otwórz okno `Animator`.
2. W lewym górnym rogu znajdź zakładkę `Parameters`.
3. Kliknij znak `+`.
4. Wybierz `Trigger`.
5. Nadaj mu nazwę: `PlayAnimation`.

Ta nazwa musi być zgodna z nazwą używaną w skrypcie `ConfigurablePart`.



Rys. 8: Dodawanie parametru `Trigger` o nazwie `PlayAnimation`.

Utworzenie przejścia animacji

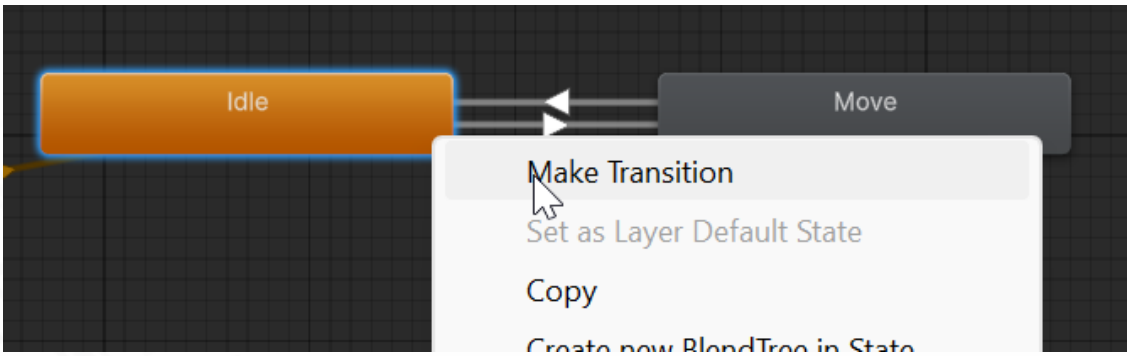
Jeżeli w kontrolerze są dwa stany, na przykład: `Idle` — stan spoczynkowy, `PartClickAnimation` — animacja kliknięcia, to trzeba utworzyć przejście między nimi.

Krok po kroku:

1. Kliknij prawym przyciskiem na stanie `Idle`.
2. Wybierz `Make Transition`.
3. Kliknij stan `PartClickAnimation`.
4. Zaznacz strzałkę przejścia.
5. W `Inspectorze` znajdź sekcję `Conditions`.
6. Dodaj warunek oparty o trigger `PlayAnimation` klikając +.

W prostszej wersji ćwiczenia można też użyć pojedynczego stanu animacji, ale wersja z przejściem lepiej pokazuje działanie animatora.

Tę linię dopisz w pliku `ConfiguratorManager.cs`, wewnątrz metody `CheckClick()`, po wywołaniu `selectedPart.Select()`.



Rys. 9: Przejście między stanami i warunek `Trigger`.

Listing 11: Wstawka: uruchamianie animacji w ConfiguratorManager

```
// Wywołanie animacji na zaznaczonej części
selectedPart.PlayClickAnimation();
```

To wywołanie integruje menedżer kliknięcia z logiką animacji części. Dzięki temu efekt animacji uruchamia się dokładnie w momencie wyboru obiektu. w pełnym przebiegu zostanie wykonane obok zaznaczenia i odtworzenia dźwięku.

ETAP 2E — dodanie dźwięku

Tę wstawkę dopisz w pliku `ConfigurablePart.cs`, wewnątrz klasy `ConfigurablePart`, obok obsługi animacji.

Listing 12: Wstawka: pola i metoda dźwięku w ConfigurablePart

```
// Ustawienia dźwięku kliknięcia
[Header("Dźwięk")]
[SerializeField] private AudioSource audioSource;
[SerializeField] private AudioClip clickSound;

public void PlayClickSound()
{
    // Brak AudioSource - brak odtwarzania
    if (audioSource == null) return;
    // Brak klipu dźwiękowego - brak odtwarzania
    if (clickSound == null) return;
    // Odtworzenie krótkiego dźwięku kliknięcia
    audioSource.PlayOneShot(clickSound);
}
```

Dodaje obsługę dźwięku poprzez pola `AudioSource` i `AudioClip` oraz metodę `PlayClickSound()`. Zapewnia akustyczne potwierdzenie kliknięcia i poprawia czytelność interakcji użytkownika. każda część może mieć inny klip, a metoda jest zabezpieczona przed brakującymi referencjami.

Tę linię dopisz w pliku `ConfiguratorManager.cs`, wewnątrz metody `CheckClick()`, po wywołaniu animacji.

Listing 13: Wstawka: uruchamianie dźwięku w ConfiguratorManager

```
// Wywołanie dźwięku na zaznaczonej części
selectedPart.PlayClickSound();
```

To końcowe spięcie menedżera z logiką audio obiektu. Powoduje odtworzenie dźwięku po kliknięciu aktualnie wybranej części. razem z wywołaniem animacji buduje pełną odpowiedź obiektu na interakcję myszy.

Dodanie komponentu AudioSource

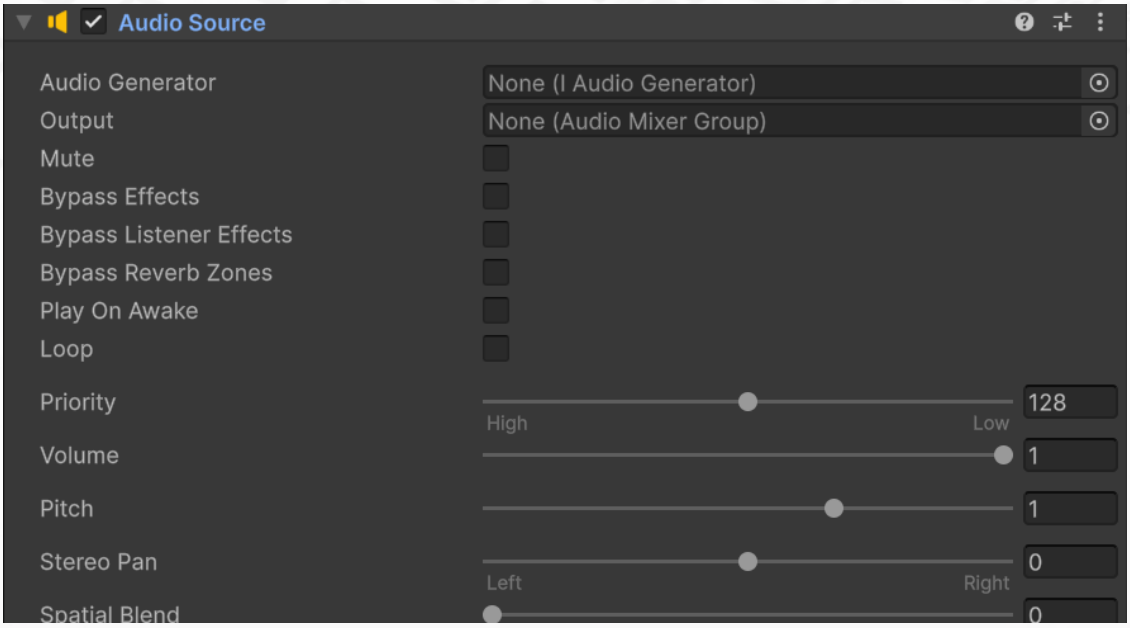
Aby obiekt mógł odtworzyć dźwięk, trzeba dodać komponent `AudioSource`.

Krok po kroku:

1. Zaznacz obiekt w `Hierarchy`.
2. W `Inspectorze` kliknij `Add Component`.
3. Wpisz `AudioSource`.
4. Dodaj komponent.

Na potrzeby tego ćwiczenia zaleca się ustawić: `Play On Awake` — wyłączone, `Loop` — wyłączone, `Spatial Blend` — 0, jeżeli dźwięk ma być prostym dźwiękiem 2D.

Dzięki temu dźwięk nie uruchomi się sam po starcie sceny i nie będzie zapętlany.



Rys. 10: Komponent `AudioSource` z podstawowymi ustawieniami.

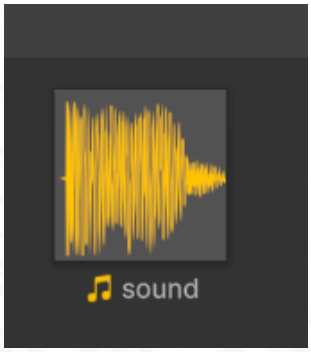
Import pliku dźwiękowego

Do odtworzenia dźwięku potrzebny jest plik audio, na przykład: `.wav`, `.ogg`, `.mp3`.

Krok po kroku:

1. Przeciągnij plik dźwiękowy do okna `Project`, albo kliknij prawym przyciskiem w oknie `Project` i dodaj plik do folderu projektu z poziomu systemu operacyjnego.
2. Po zaimportowaniu plik będzie widoczny jako `AudioClip`.

Wskazówka: na potrzeby ćwiczenia najlepiej użyć krótkiego dźwięku, na przykład: kliknięcia, aktywacji, mechanicznego ruchu, przesunięcia elementu.



Rys. 11: Zaimportowany plik dźwiękowy w oknie `Project`.

Przypisanie dźwięku do skryptu

W skrypcie `ConfigurablePart` znajdują się pola: `audioSource`, `clickSound`. Dlatego w `Inspectorze` trzeba przypisać odpowiednie obiekty.

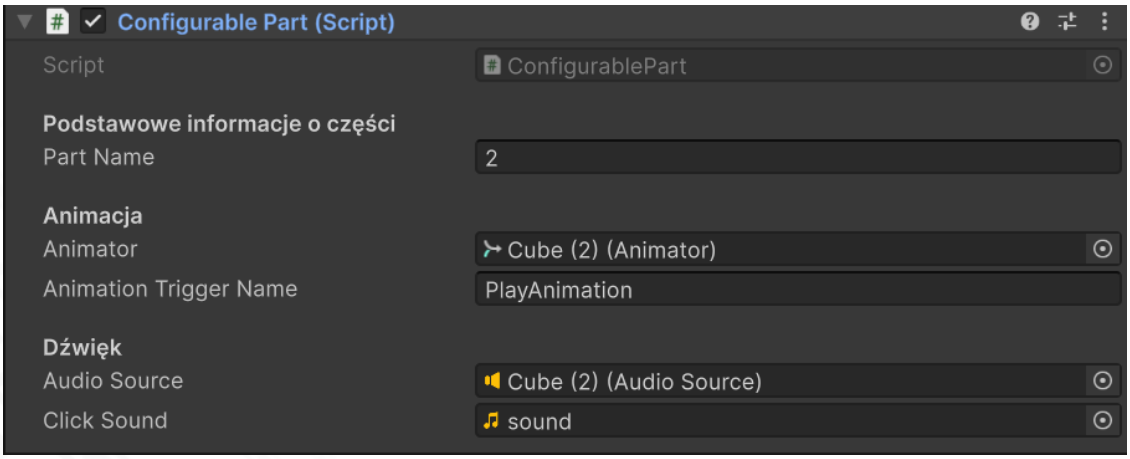
Krok po kroku:

1. Zaznacz obiekt z komponentem `ConfigurablePart`.
2. W polu `Audio Source` przeciągnij komponent `AudioSource`.
3. W polu `Click Sound` przeciągnij plik audio z okna `Project`.

Podobnie dla animacji:

1. W polu `Animator` można przeciągnąć komponent `Animator`.
2. W polu `Animation Trigger Name` pozostawić nazwę: `PlayAnimation`.

Ważne: Powyższe wstawki służą tylko do kopiowania krok po kroku. Pełne skrypty do pobrania znajdują się w rozdziale 5.



Rys. 12: Pola `Animator`, `Audio Source` i `Click Sound` w komponencie `ConfigurablePart`.

4..4 Dodatkowe wskazówki implementacyjne

Poniższe wskazówki rozszerzają instrukcję i pomagają uniknąć najczęstszych błędów podczas samodzielnej pracy.

Kolejność konfiguracji w `Inspector`

1. Upewnij się, że każdy obiekt klikalny ma `Collider` oraz komponent `ConfigurablePart`.
2. Sprawdź, czy `Main Camera` ma przypięty skrypt `CameraOrbit` oraz poprawnie ustawione pole `Target`.
3. W obiekcie menedżera ustaw `Main Camera`, `Clickable Layer` i odpowiedni zasięg promienia `Max Ray Distance`.
4. Dla części z animacją przypisz `Animator` i zweryfikuj nazwę triggera `PlayAnimation`.
5. Dla części z dźwiękiem przypisz `AudioSource` i plik `Click Sound`.

Szybka checklista testów w `Play mode`

- prawy przycisk myszy obraca kamerę wokół obiektu,
- rolka myszy zmienia odległość kamery w zakresie `Min Distance -- Max Distance`,
- najechanie kursorem zmienia stan podświetlenia części,
- kliknięcie lewym przyciskiem zaznacza tylko jeden obiekt,
- kliknięcie uruchamia animację i dźwięk dla wybranej części,
- kliknięcie poza obiektem czyści zaznaczenie.

Typowe problemy i szybka diagnoza

- **Brak reakcji na kliknięcie:** sprawdź `Collider`, warstwę obiektu i maskę `Clickable Layer`.
- **Brak hover:** upewnij się, że promień trafia w ten sam obiekt, który ma `ConfigurablePart` (lub jego rodzica).
- **Animacja nie startuje:** zweryfikuj nazwę triggera i przejścia w `Animator Controller`.
- **Brak dźwięku:** sprawdź przypisanie `AudioSource`, `AudioClip` oraz poziom głośności komponentu.
- **Kamera nie porusza się:** sprawdź, czy `target` nie jest pusty i czy działa `Input System`.

5. Wymagania zaliczeniowe

1. Kamera obraca się i zoomuje zgodnie z założeniami (bez odwracania osi i skoków pozycji).
2. Obiekty reagują na `hover` i `click` zgodnie z logiką jednego aktywnego zaznaczenia.
3. Kliknięcie uruchamia poprawny trigger animacji przypisany do wybranej części.
4. Kliknięcie odtwarza dźwięk przypisany do tej samej części.
5. Kliknięcie poza obiektem resetuje stan zaznaczenia.

OCENA

Student dostaje:

- ocenę **3** za prawidłowo wykonany przykład z instrukcji zgodny z wymaganiami zaliczeniowymi,
- ocenę **5** za prawidłowo wykonany własny model z conajmniej 5-ma częściami zgodny z wymaganiami zaliczeniowymi,

Uwagi końcowe

Model proszę pobrać ze strony Unity Assets Store `https://assetstore.unity.com/?category=3d%2Fprops&orderBy=4`