

POLITECHNIKA RZESZOWSKA
KATEDRA INFORMATYKI I AUTOMATYKI
DR INŻ. MATEUSZ POMIANEK

ĆWICZENIE LABORATORYJNE - UNITY 1

Środowisko · Nawigacja · Przyciski · Interakcje

Programowanie Aplikacji Mobilnych - moduł U

Spis treści

1.	Środowisko edytora unity	1
2.	Tworzenie nowego projektu	2
3.	Edycja projektu	8
4.	Dodawanie Canvas	10
5.	Dodanie tła projektu	12
6.	Tło nagłówka	14
7.	Tekst i napisy	15
8.	Przyciski i interakcje	16
9.	Nowy skrypt C#	18
10.	Zadania dla studenta	19
11.	Zadania dodatkowe	20

1. Środowisko edytora unity

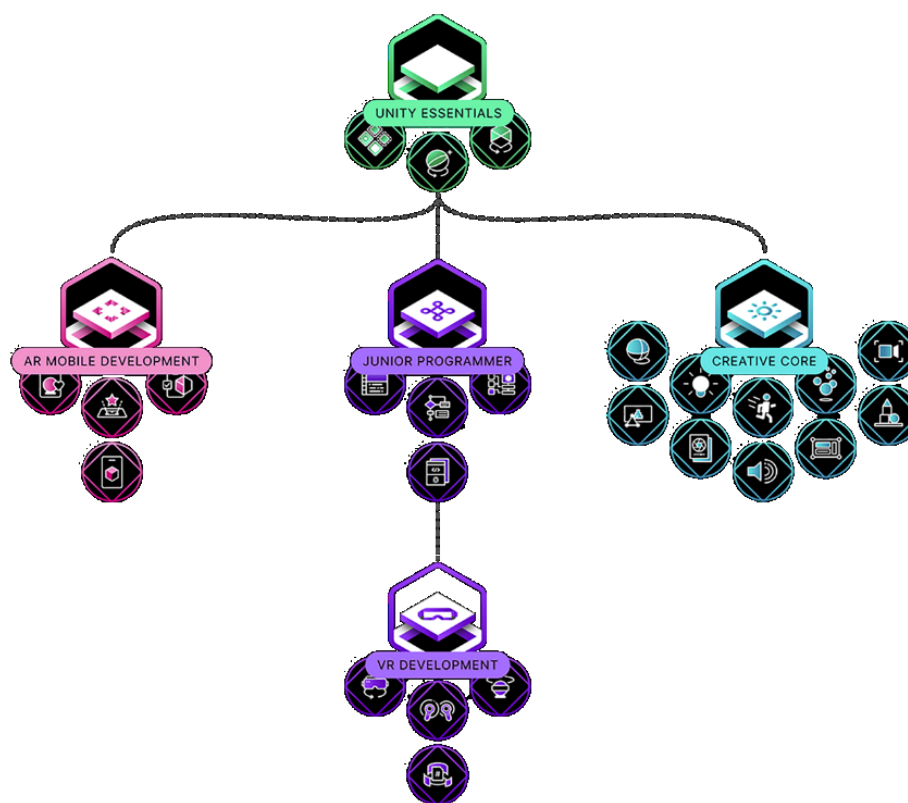
Znajdujemy ikonę oraz uruchamiamy program Unity Hub. Jeśli jest to kolejne uruchomienie swojego projektu, to edytor można otworzyć poprzez konkretną wersję środowiska.



Rys. 1. Ikony Unity Hub

Następnie logujemy się lub tworzymy nowe konto. Dla studentów Unity przygotowało dedykowaną darmową wersję - <https://unity.com/products/unity-student>.

Środowisko Unity, mimo że pierwotnie dedykowane do tworzenia gier komputerowych, obecnie jest jednym z największych edytorów aplikacji interaktywnych. Umożliwia zarówno projektowanie gier na różne platformy, ale również dowolnych aplikacji **2D**, **3D**, **AR** and **VR** dla branż.

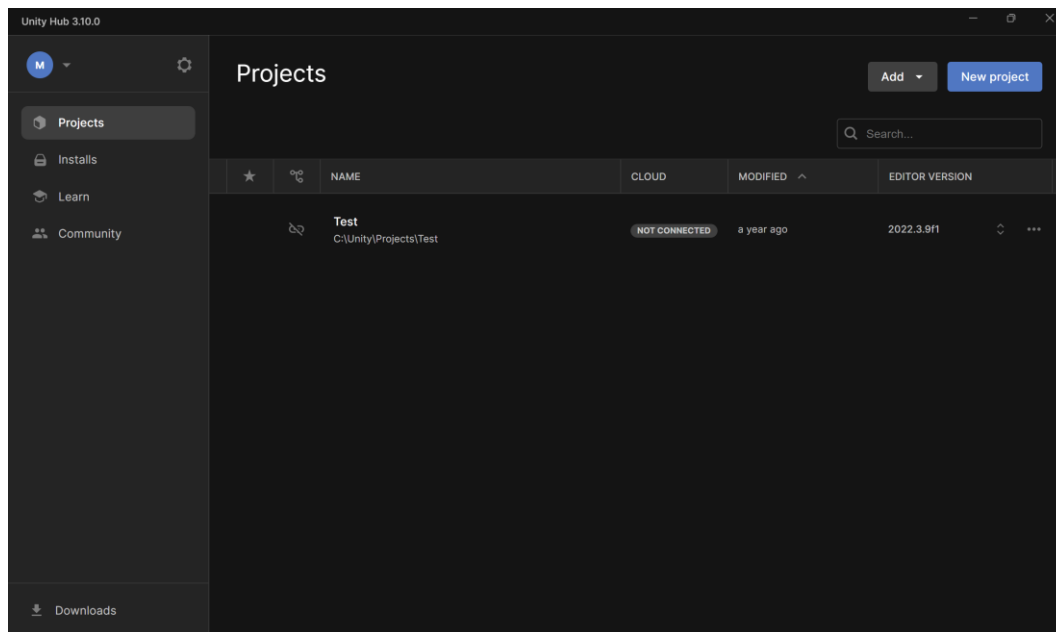


Rys. 2. Wykres poszczególnych aplikacji edytora Unity

Niezaprzeczalnym atutem tego środowiska jest społeczność oferująca wiele przydatnych narzędzi. Unity w celu zachęcenia do wykorzystywania ich środowiska oferuje wiele darmowych kursów pogrupowanych w odpowiednie kategorie zainteresowań, takich jak [Unity Essentials](#) czy [Junior Programmer](#). Więcej na ich temat można poczytać na <https://learn.unity.com/>. Uzupełnieniem omawianych na tym laboratorium zagadnień mógłby być np. kurs [Mobile Development Techniques](#) lub instrukcja [Getting started with Android](#).

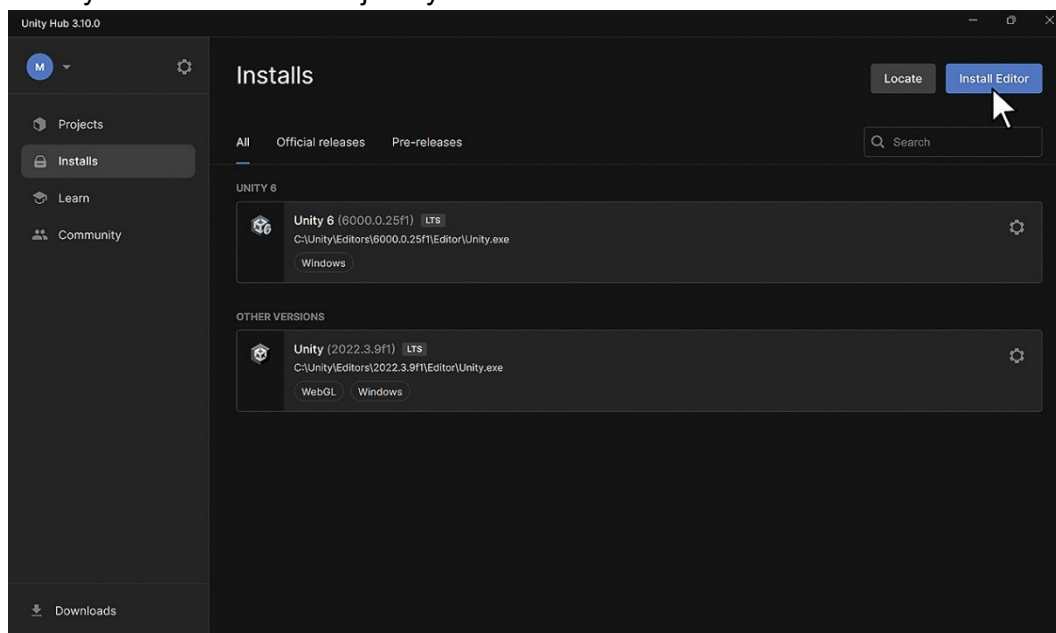
2. Tworzenie nowego projektu

Po uruchomieniu aplikacji Unity Hub ukazuje się nam okno prezentujące listę istniejących projektów (o znanej lokalizacji). Po przeniesieniu projektu z zewnętrznego źródła należy dodać jego lokalizację ręcznie, aby edytor mógł go odnaleźć i otworzyć.



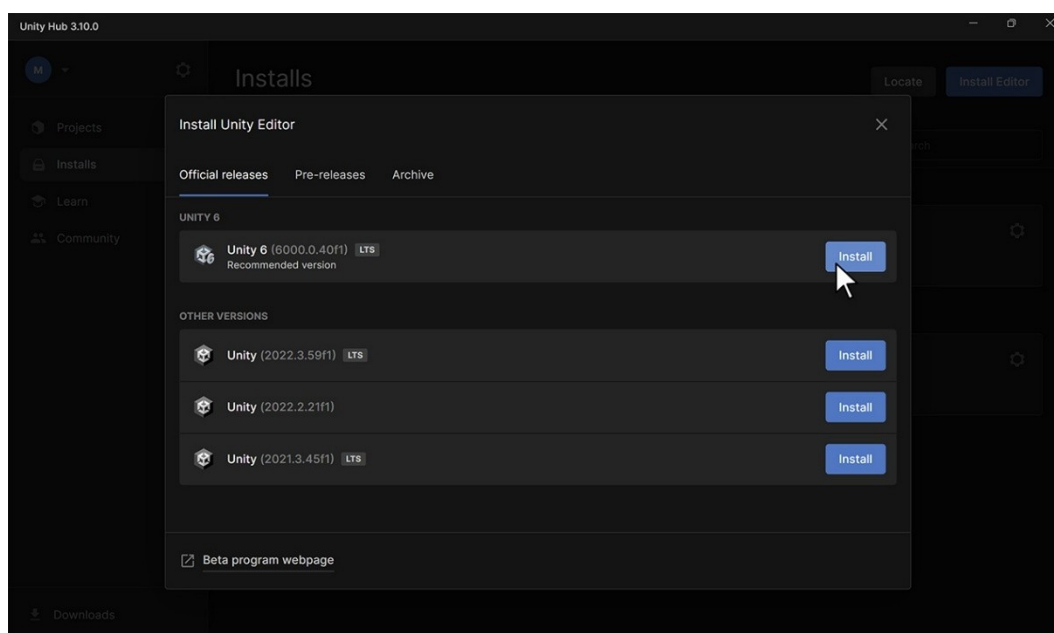
Rys. 3. Widok projektów edytora Unity

Staramy się pracować na możliwie nowej wersji środowiska, dlatego w zakładce **Installs** sprawdzamy zainstalowane wersje edytora.



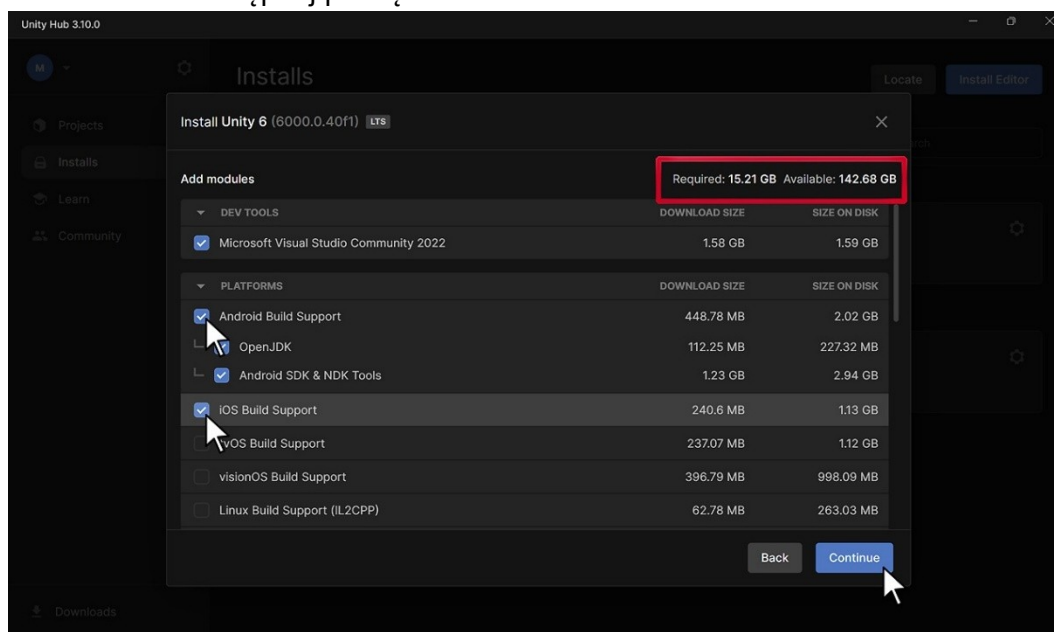
Rys. 4. Zainstalowane wersje edytora Unity

Założmy, iż satysfakcjonujące dla nas będą wersje powyżej **Unity 6 (6000.0.25f1)**. Jeżeli system posiada starszą wersję - znacznie różniącą się od aktualnej - konieczne jest zainstalowanie nowej. W tym celu naciskamy na przycisk **Install Editor**.



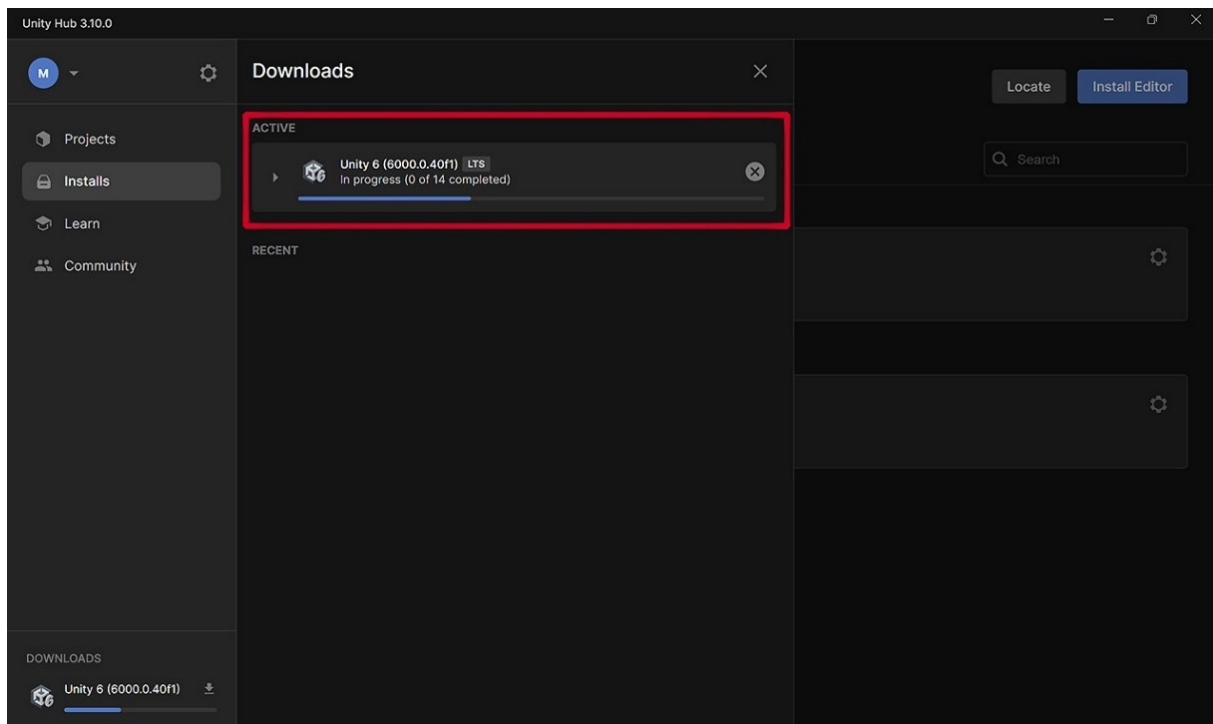
Rys. 5. Wybór wersji edytora do zainstalowania

Konieczne jest również określenie pakietów, które zostaną zainstalowane wraz z nową wersją edytora. Chcąc tworzyć aplikację dla rozwiązań 2D racjonalnym będzie więc dodać pakiety dla Android oraz iOS. Zwracamy przy okazji uwagę na wzrost ilości zajmowanego miejsca i to, czy nie przekracza ono dostępnej pamięci.



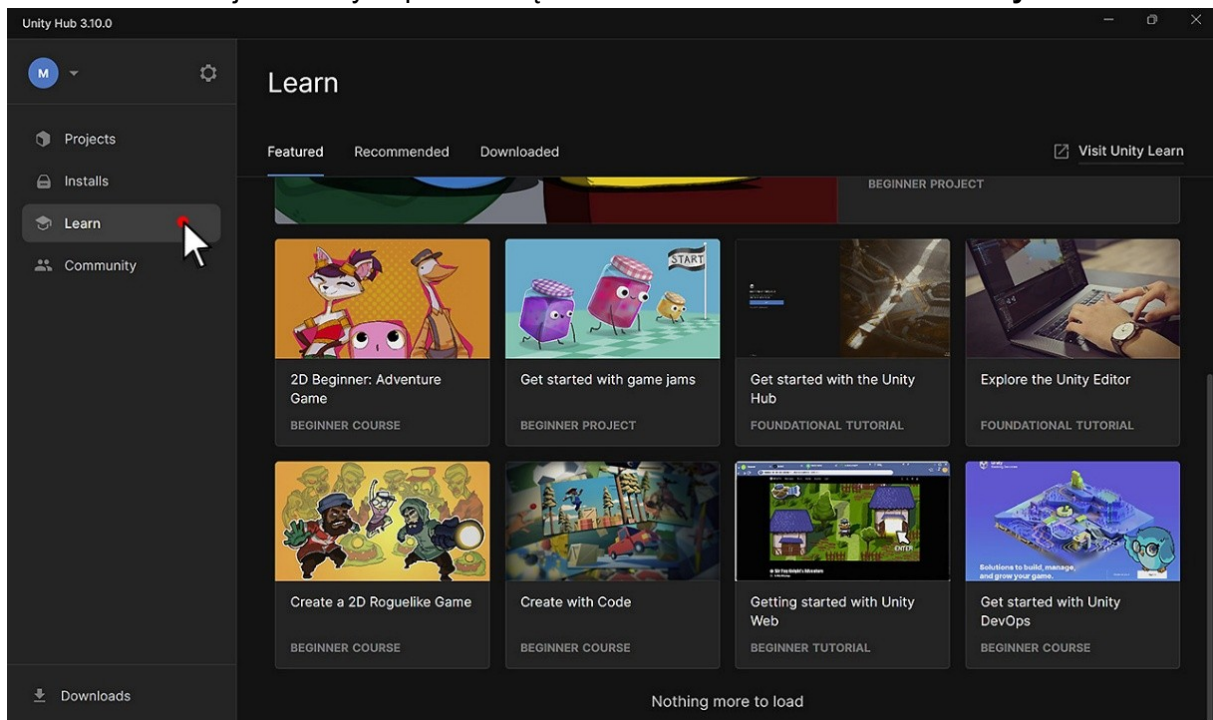
Rys. 6. Wybór dodatkowych pakietów instalowanych wraz z nową wersją edytora Unity

Następnie akceptujemy wszystkie kolejne zgody i oczekujemy na zakończenie instalacji.

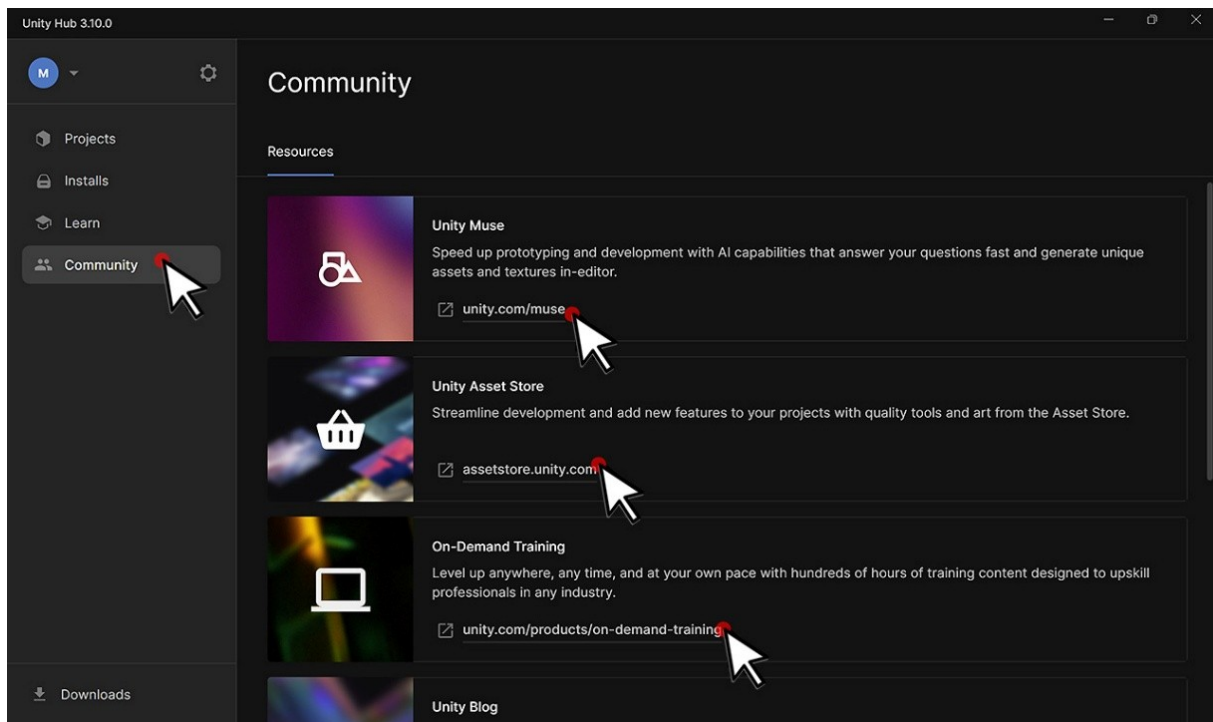


Rys. 7. Okno instalacji edytora Unity

W czasie instalacji możemy zapoznać się z zakładkami **Learn** oraz **Comiunity**.



Rys. 8. Zakładka Learn



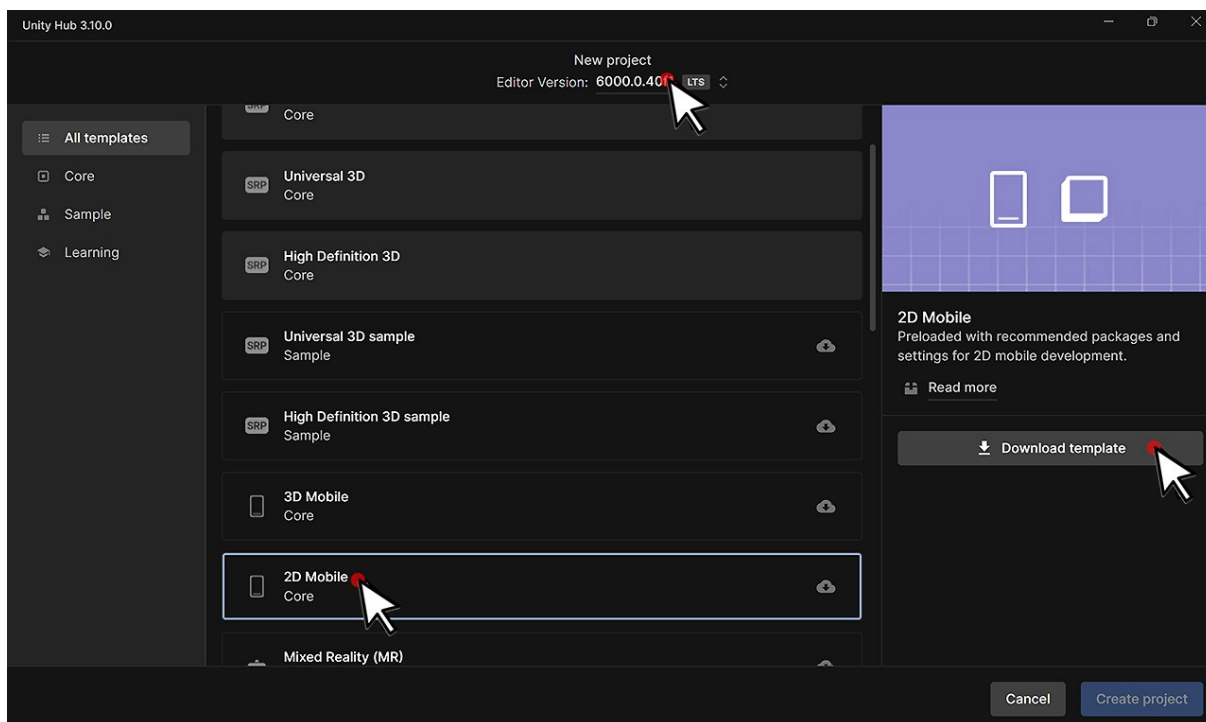
Rys. 9. Zakładka Comunity

W zakładce **Learn** znajdują się ciekawe kursy i przykładowe projekty poruszające szerszy zakres budowania aplikacji niż opisany w tej instrukcji. Przykładowymi szkoleniami, mogą być:

1. [Setting Up the Project](#)
2. [Memory Management in Unity](#)
3. [Introduction to Editor Scripting](#)
4. [UI Masking](#)
5. [Platformer Microgame](#)
6. [Performance and optimization](#)
7. [Unity Version Control: Quick start guide](#)
8. [Video: 2D Game Kit Walkthrough](#)

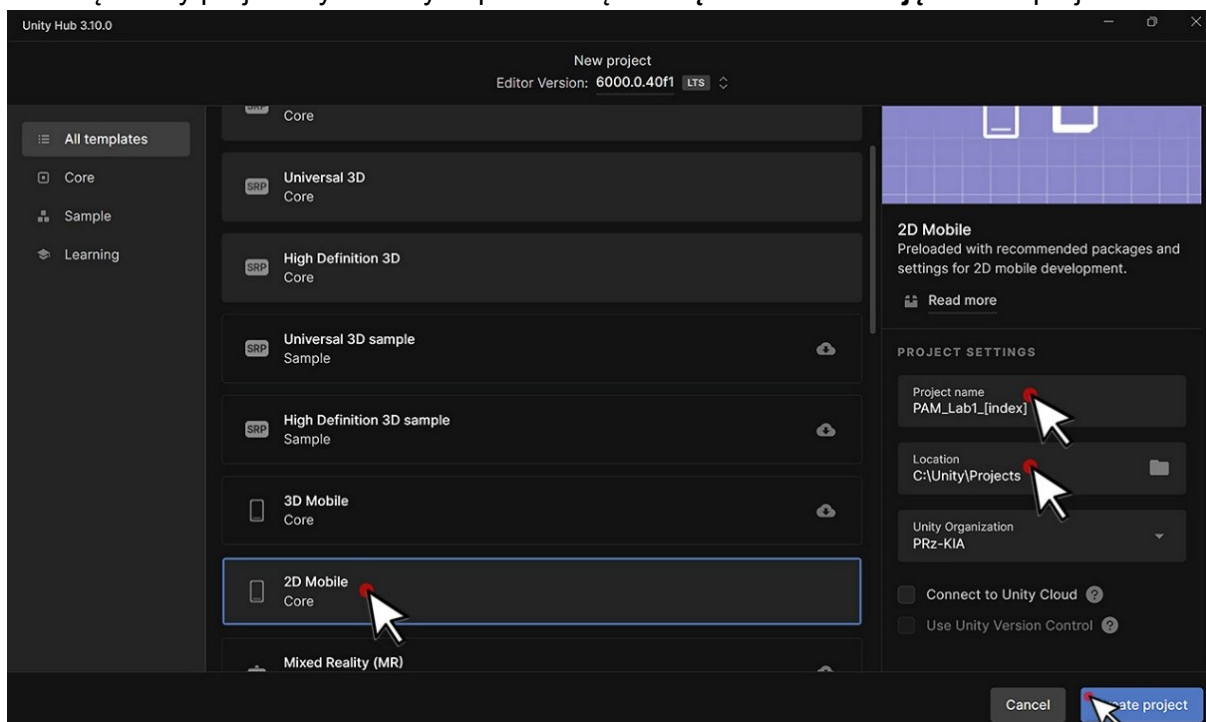
W zakładce **Community** znajdują się z kolei odnośniki do zewnętrznych baz danych, **zasobów** oraz rozwiązań komercyjnych oferowanych przez społeczność Unity. Szczególną popularnością cieszy się [Asset Store](#), na którym dostępne są liczne płatne i darmowe zasoby do użycia we własnych projektach.

Gdy zainstalujemy interesującą nas wersję edytora Unity naciskamy przycisk **nowy projekt**, a następnie wybieramy **2D Mobile** oraz pobieramy **Template** dotyczący tego typu projektu. Należy upewnić się, czy wybrany rodzaj projektu tworzymy w pożądanej wersji edytora. Jego numer widzimy w górnej części okna.



Rys. 10. Wybór rodzaju projektu

Tworząc nowy projekt wybieramy odpowiednią **nazwę** oraz **lokalizację** folderu projektu.

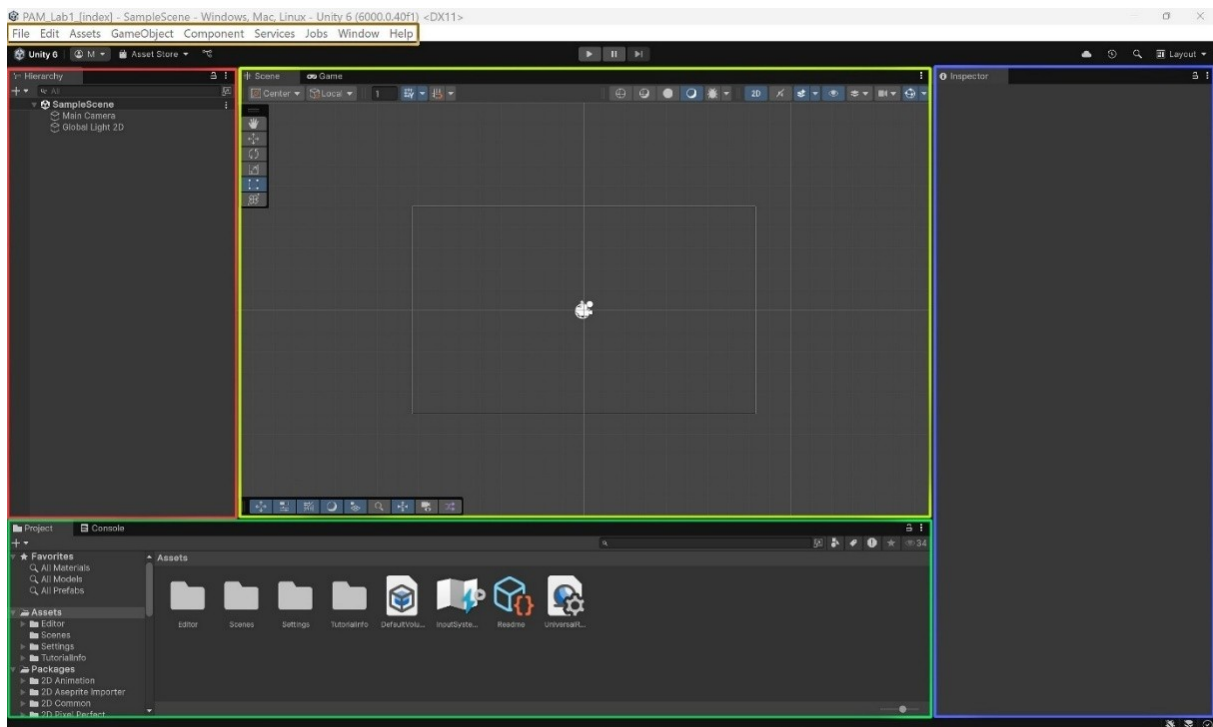


Rys. 11. Wybór nazwy i lokalizacji

3. Edycja projektu

Po otwarciu wybranego pliku naszym oczom ukazuje się edytor projektu. W górnej części okna znajduje się klasyczny pasek zakładek. Poniżej edytor zawiera okna w wybranej przez użytkownika konfiguracji. Domyślnie widok ten składa się z:

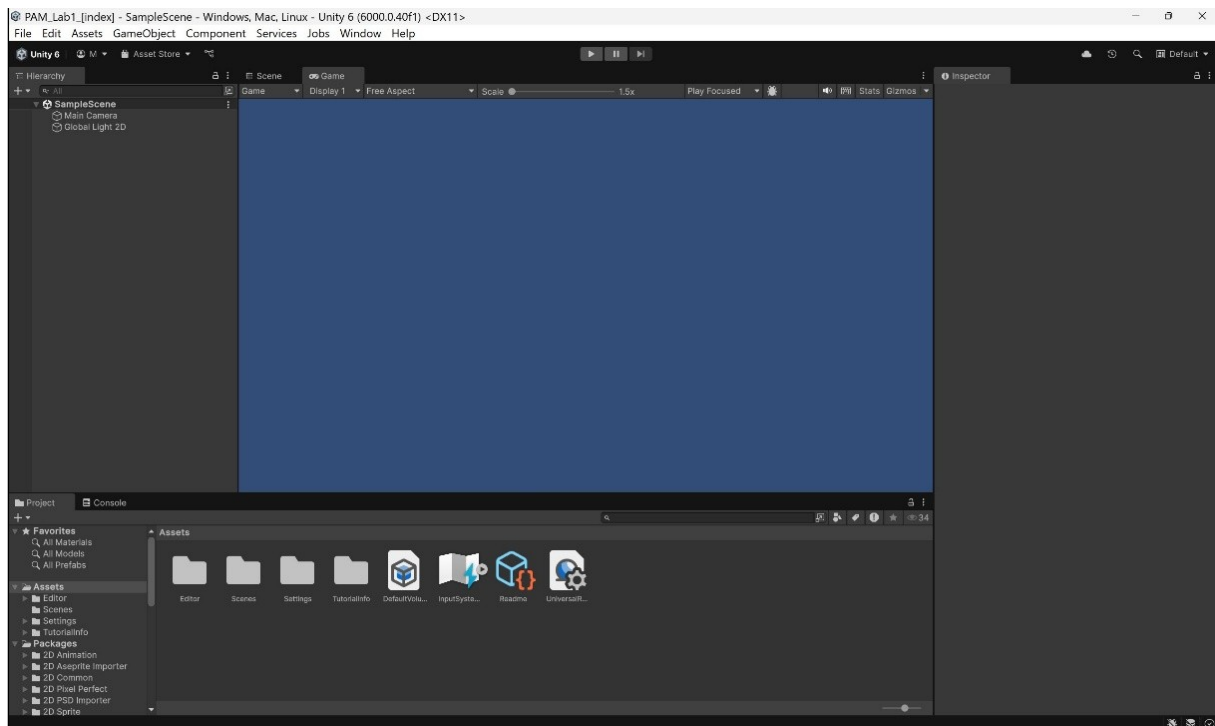
1. Hierarchii obiektów w scenie
2. Okna edycji sceny
3. Okno plików projektu / terminal
4. Inspektora właściwości wybranego obiektu



Rys. 12. Okno edytora Unity

Oprócz opisanych na rysunku okien występuje jeszcze jedno kluczowe - Game. Przełączenie się na to okno występuje w momencie uruchomienia aplikacji poprzez naciśnięcie strzałki znajdującej się w górnej części edytora.

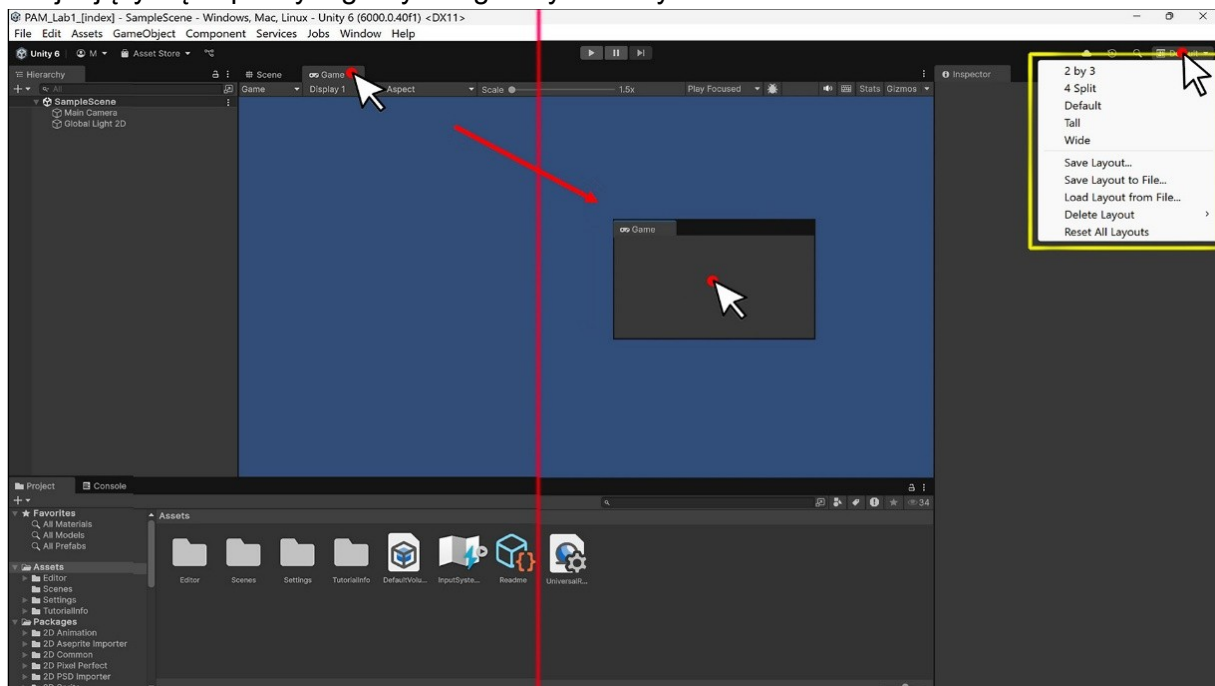
Należy mieć na uwadze, iż gdy aplikacja jest uruchomiona, to jakiegokolwiek zmiany wprowadzone w tym czasie nie zostaną zapisane w projekcie!



Rys. 13. Okno Game

Można także ręcznie przełączyć się na to okno poprzez naciśnięcie przycisku **Game** znajdującego się powyżej centralnego okna edycji obok **Scene**. Pozycje okien można dostosować zależnie od własnych preferencji i wygody.

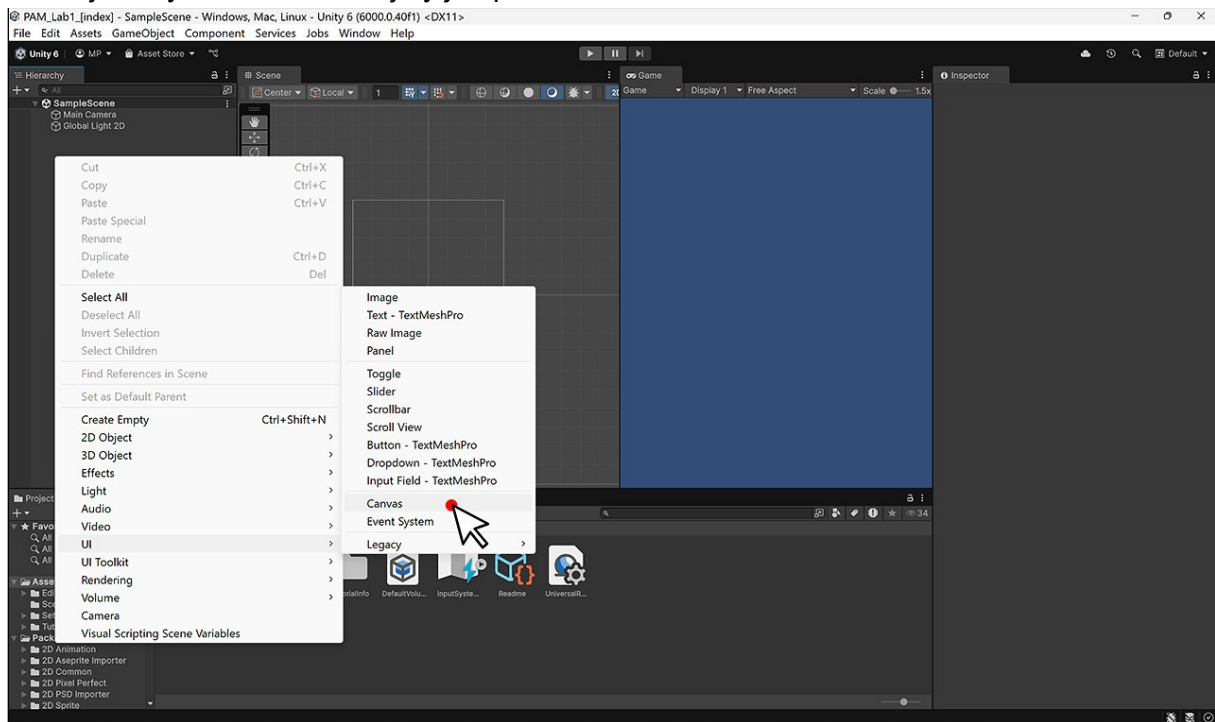
Istnieje również możliwość ręcznego przeciągnięcia okna GAME, tak aby pokazywało ono wygląd projektu. Aby przywrócić wygląd domyślny wystarczy nacisnąć przycisk **Layout** znajdujący się w prawym górnym rogu edytora i wybrać widok **Default**.



Rys. 14. Zmiana pozycji okien edytora

4. Dodawanie Canvas

Tworzymy nowy obiekt - **Canvas** - stanowiący „płótno”, na które będą nanoszone elementy interfejsu użytkownika. Nazwijmy je np. **Canvas 1**.

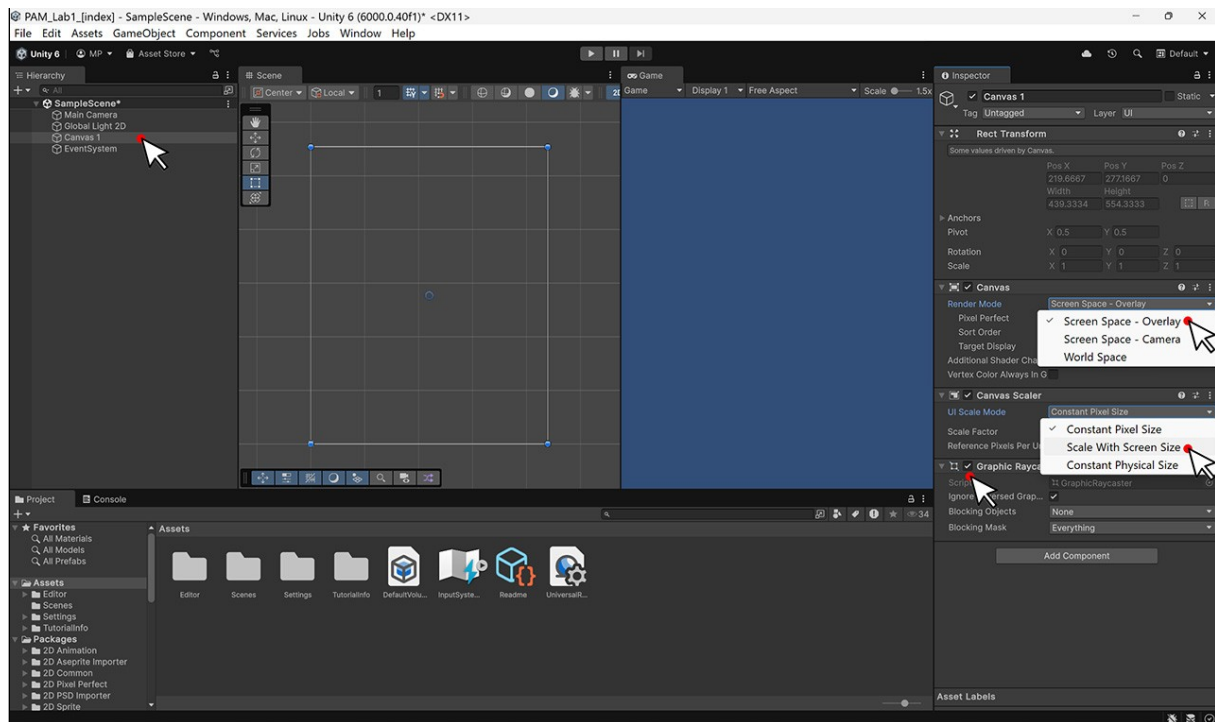


Rys. 15. Dodawanie obiektu Canvas

Następnie przechodzimy do edycji parametrów naszego obiektu. Jego komponenty (wyświetlone w oknie **Inspektora** po prawej stronie) określają właściwości w poszczególnych kategoriach. I tak parametry **Render Mode** determinuje w odniesieniu do jakiej przestrzeni wyświetlany będzie obiekt. Pierwsza opcja stanowi nakładkę na ekran aplikacji, czyli to co znajduje się w polu widzenia kamery. Druga dopasuje go do widoku wirtualnej kamery względem jej ustawień. Trzecia natomiast umieszcza obiekt płótna w przestrzeni wirtualnej nadając mu dedykowaną pozycję oraz wymiary. Pozostawmy domyślną opcję **Screen Space - Overlay**. Przy okazji zaznaczymy parametry **Pixel Perfect**. Wyostrzy to wygląd elementów umieszczonych wewnątrz tego Canvas, chociaż kosztem płynności potencjalnych animacji. Jednak z racji iż planujemy utworzyć nieruchomy obraz, to zaznaczenie tej opcji będzie dobrym wyborem.

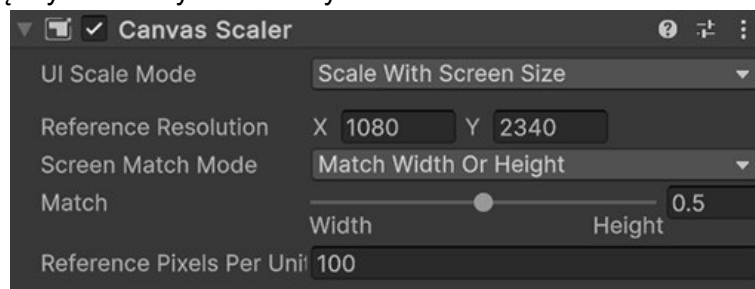
Zwracamy również uwagę na komponent o nazwie **Graphic Raycaster**. Określa on w jaki sposób obiekty wewnątrz naszego **Canvas** mają reagować na interakcje z myszą. Wyłączenie tego komponentu uniemożliwi wykrywanie takich interakcji, jednak oszczędzi zasoby obliczeniowe aplikacji, gdyż nie będzie ona musiała w każdej klatce sprawdzać wystąpienia takiego zdarzenia. Dlatego jego dezaktywacja bywa dobrym wyborem dla **Canvas** zawierające obiekty nieinteraktywne, np. obrazy tła. W naszym przypadku pozostawimy jednak ten komponent włączony, gdyż planujemy dodać przycisk. Dlatego konieczne będzie aktywne wykrywanie potencjalnych interakcji.

- Więcej na temat klasy `GameObject` można przeczytać w artykule [GameObject](#).



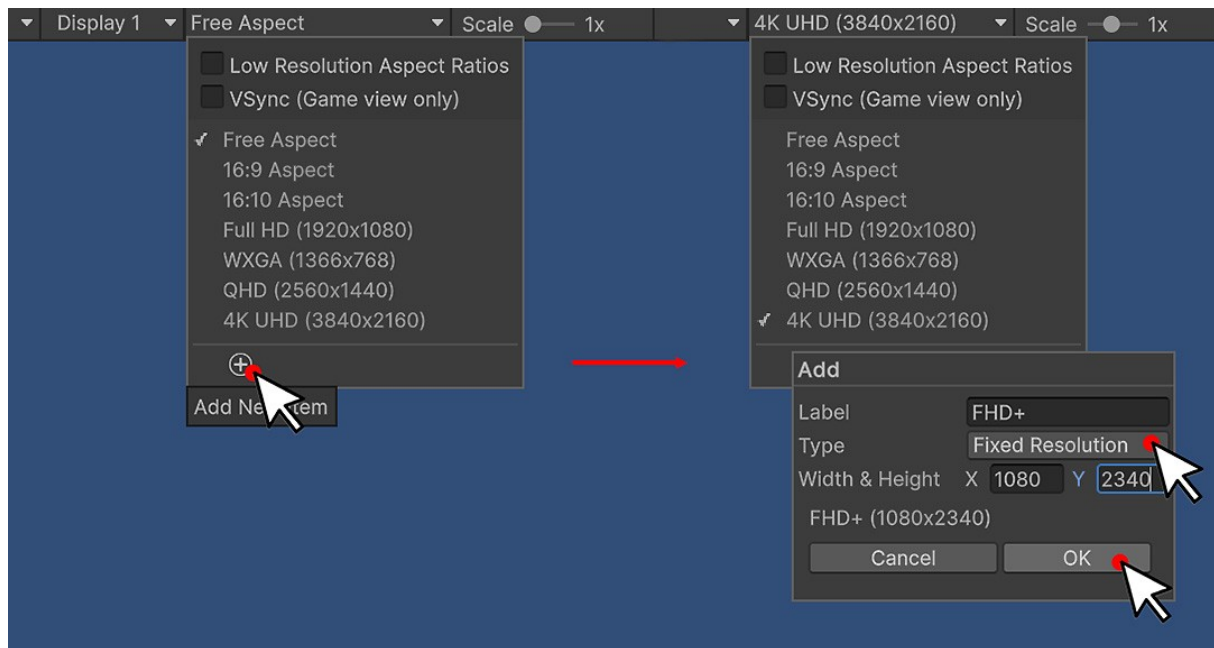
Rys. 16. Parametry obiektu Canvas

Następnie dostosowujemy ustawienia dotyczące skalowania się interfejsu przy zmianie rozdzielczości urządzenia. Określone jest to zmienną **UI Scale Mode**. Domyślnie aplikacja będzie miała stały rozmiar swojego interfejsu (w pikselach). My jednak - iż planujemy aby nasza aplikacja wyświetlała się prawidłowo na wielu urządzeniach - chcemy aby dopasowywała UI do wymiarów ekranu. Dlatego zaznaczamy opcję **Scale With Screen Size**. Określamy domyślną rozdzielczość oraz decydujemy, w jakim stopniu ma się ona skalować względem zmiany szerokości lub wysokości obrazu. Ustawienie 0,5 będzie stanowić kompromis pomiędzy oboma tymi zmiennymi.



Rys. 17. Ustawienia skalowania UI

Określiśmy jakie wymiary ma nasz obiekt **Canvas 1**. Chcemy teraz aby podgląd wyświetlany w oknie **Game** również prezentował widok odpowiadający tej rozdzielczości. Nie zawsze muszą być one jednak zgodne. Możemy wypróbować inne proporcje wyświetlaczy i przetestować jak utworzony przez nas obiekt radzi sobie ze skalowaniem. Ostatecznie ustawiamy taką samą rozdzielczość jaką zostawiliśmy dla naszego obiektu. Zakładamy, iż będzie ona reprezentowała domyślny dla nas wyświetlacz. Zamiast rozdzielczości możemy określić też same proporcje ekranu - **Aspect Ratio**.

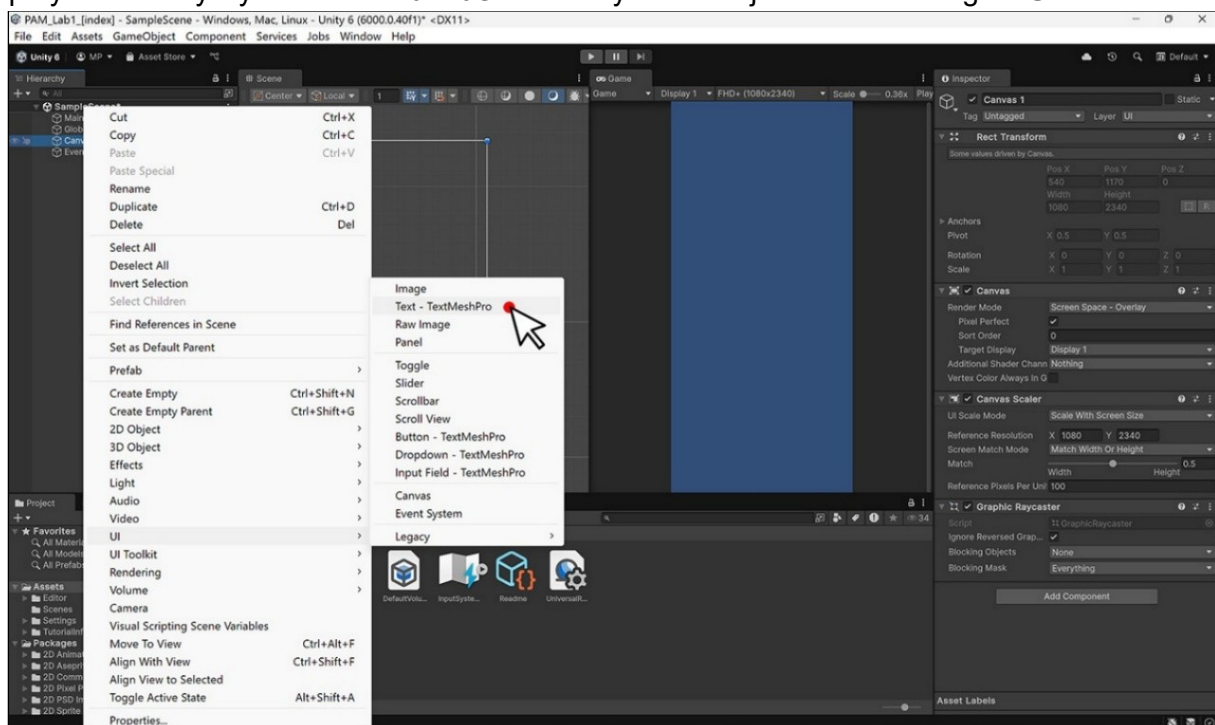


Rys. 18. Określenie wymiarów okna Game

Dwukrotne naciśnięcie na obiekt **Canvas 1**, znajdujący się w widoku **Hierarchii** po lewej stronie powinien wyśrodkować na nim widok edytora.

5. Dodanie tła projektu

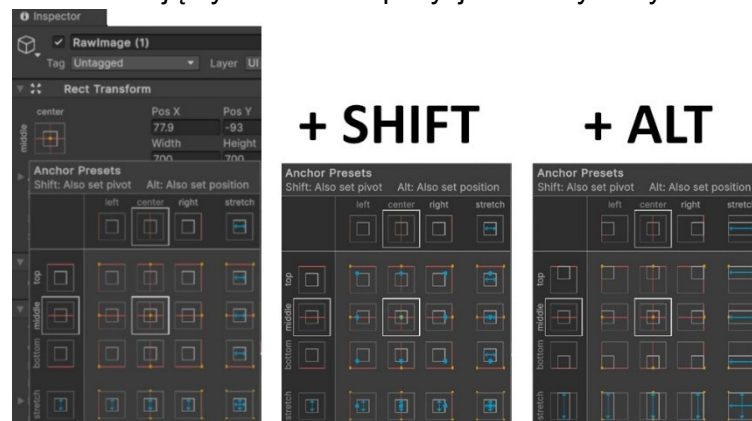
Dodajemy dwa obiekty typu **RawImage**. Wykonujemy to poprzez naciśnięcie prawym przyciskiem myszy na obiekt **Canvas 1** oraz wybór rodzaju obiektu z kategorii **UI**.



Rys. 19. Nowe obiekty GUI

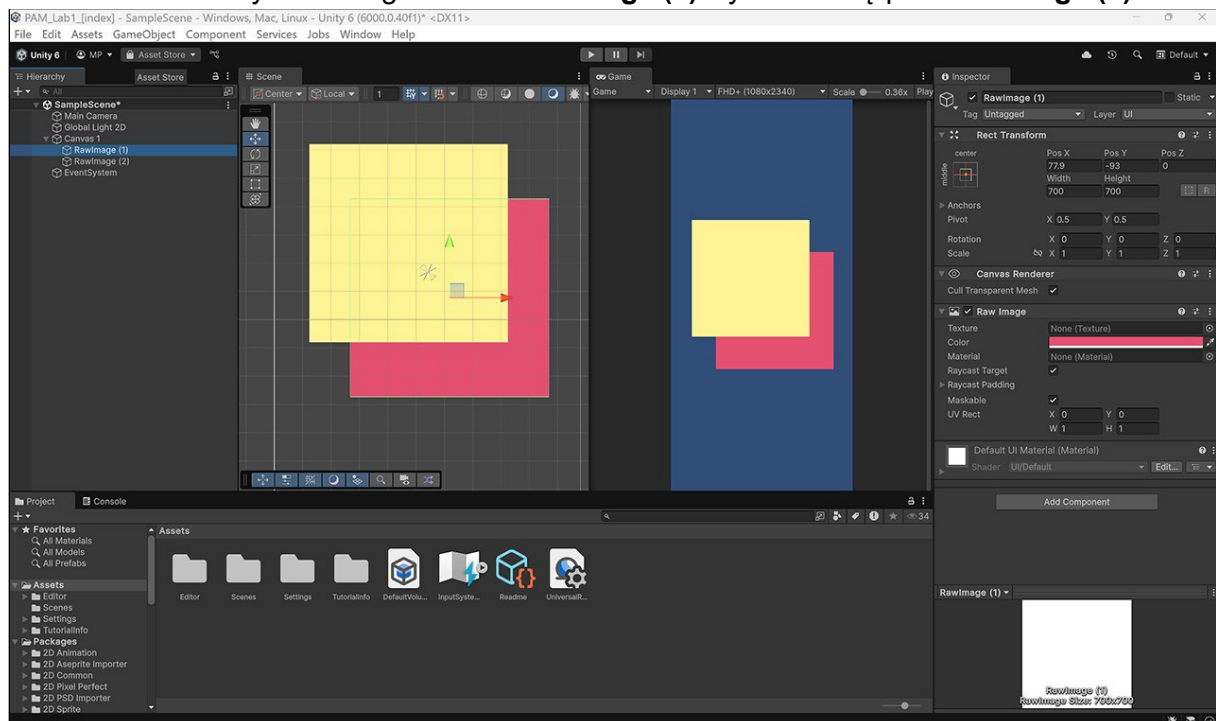
Podobnie jak w przypadku obiektu **Canvas**, tak i tu komponenty określają właściwości obiektów.

Edycja komponentu **RawImage** pozwoli np. na zmianę koloru obrazu. a edycja **Rect Transform** na dostosowanie pozycji, rozmiaru oraz „zakotwiczenia” obiektu. Determinuje to punkt (lub wymiar) względem którego określane będzie położenie oraz wymiary obiektu. Z racji, iż chcemy uczynić ten obiekt tłem, to wybieramy dostosowanie go do całego ekranu, czyli ikonę znajdującą się w prawym dolnym rogu. Opcję tą określamy wraz z klawiszami Shift (**set pivot**) oraz Alt (**set position**) aby zakotwiczyć go we wszystkich zakresach. Decydujemy względem której wartości mają być określane pozycja oraz wymiary obiektu



Rys. 20. Ustawienia kotwiczenia obiektu 2D

Obiekty w oknie **Hierarchii** ustawione są - jak sama nazwa wskazuje - hierarchicznie. Im obiekt znajduje się niżej na liście, tym na wyższej warstwie leży. Wyższe warstwy leżą ponad obiektami na niższych. Dlatego obiekt **RawImage (1)** wyświetla się pod **RawImage (2)**.

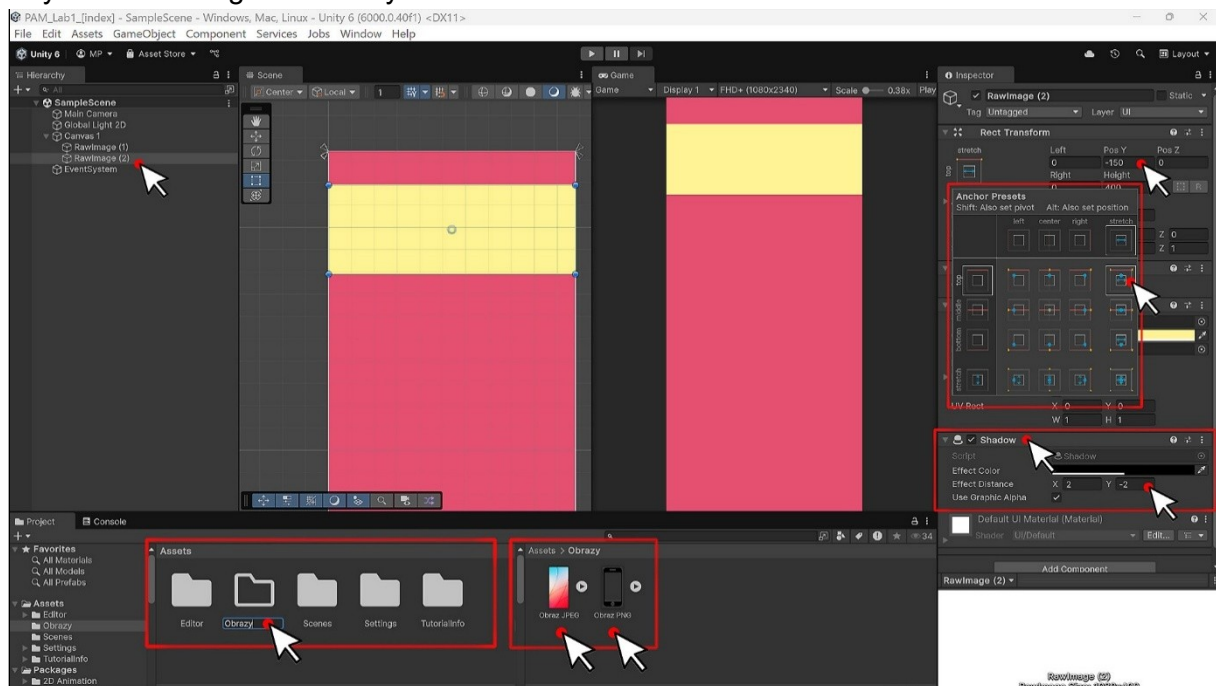


Rys. 21. Warstwy interfejsu

- Więcej na temat importu plików do projektu można przeczytać w artykule [Importing Assets](#).
- Więcej na temat klas obiektów można przeczytać w artykule [UnityEngine Objects](#).

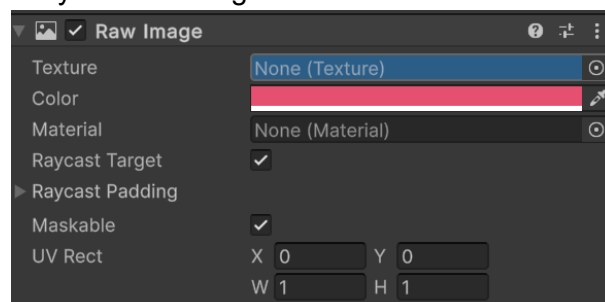
6. Tło nagłówka

Dopasowujemy obiekt **RawImage (2)** tak, aby stanowiła nasze tło nagłówka. W tym celu odpowiednio modyfikujemy jego ustawienia zakotwiczenia, pozycję oraz wymiary. Możemy również dodać do niego dodatkowy komponent - **Shadow**. Odpowiada on za efekt rzucania cienia przez obiekt. Możemy poeksperymentować z poszczególnymi parametrami cienia tak, aby dostosować go do własnych oczekiwań.



Rys. 22. Dostosowywanie pozycji i komponentów obiektu

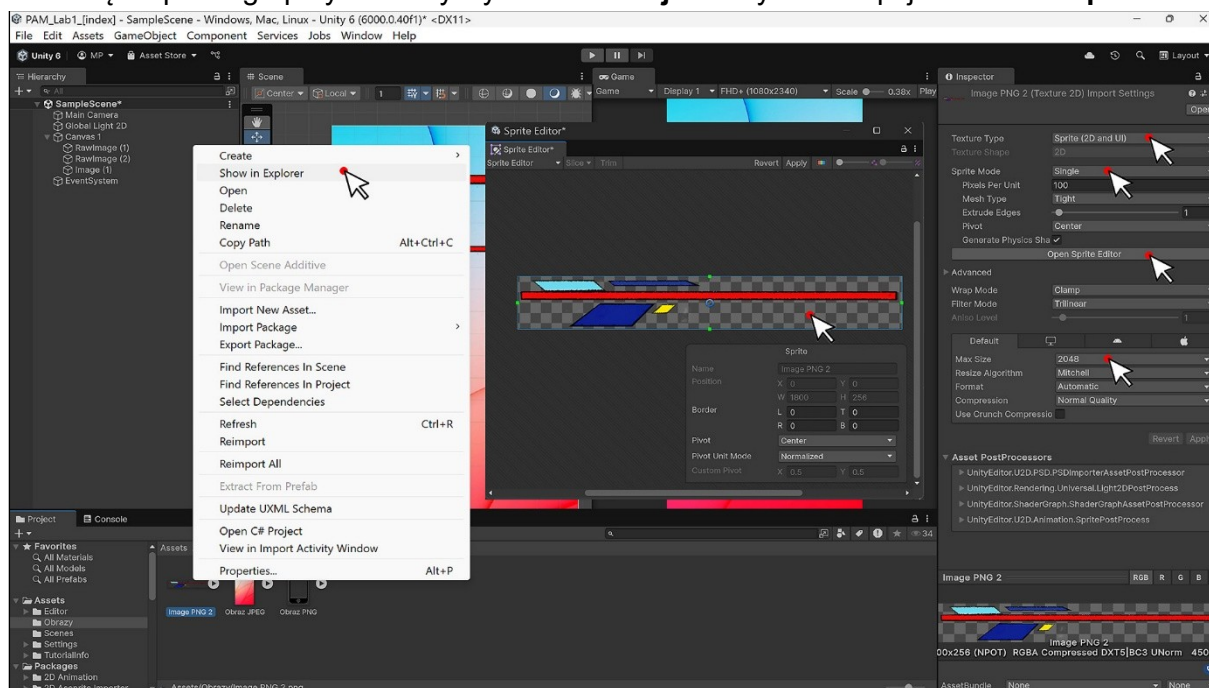
W celu urozmaicenia graficznego naszej aplikacji, tworzymy nowy folder o nazwie **Obrazy**. Pobieramy z Internetu dwie grafiki. Pierwsza będzie tłem, dlatego wystarczy nam obraz skompresowany **JPEG**. Drugi chcemy zastosować aby zaprezentować efekt przeźroczystości obrazu, dlatego konieczny będzie kanał alfa obrazu. W tym celu pobieramy taki obraz w formacie **PNG**. Wybrany obraz przypisujemy jako teksturę obiektu **RawImage (1)** w polu **Texture**. Podobnie możemy zrobić z drugim obiektem.



Rys. 23. Ustawienia obiektu RawImage

Tworzymy kolejny obiekt, tym razem typu **Image**. Spróbujemy przypisać do niego nasz drugi obraz jako teksturę. Zauważymy jednak, iż obraz nie jest w tej formie akceptowalny, gdyż pole tekstury oczekuje obiektu typu **Sprite**. Dlatego jesteśmy zmuszeni edytować opcje naszego obrazu PNG. Podczas edycji zwróćmy również uwagę na inne jego parametry i dostosujmy je do własnych potrzeb. W razie konieczności, obiekty typu **Sprite** można edytować również poprzez uruchomienie okna **Sprite Editor**.

Folder w którym umieściliśmy nasze obrazy można szybko otworzyć w eksploratorze poprzez naciśnięcie prawego przycisku myszy w oknie **Project** i wybraniu opcji **Show in Explorer**.

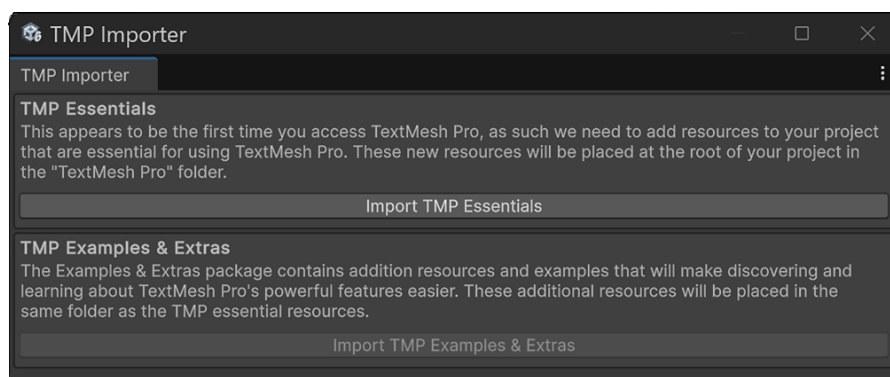


Rys. 24. Edycja ustawień obrazu

- Więcej na temat tekstur można znaleźć w artykule [Get started with textures](#).
- Więcej na temat materiałów można znaleźć w artykule [Materials](#).

7. Tekst i napisy

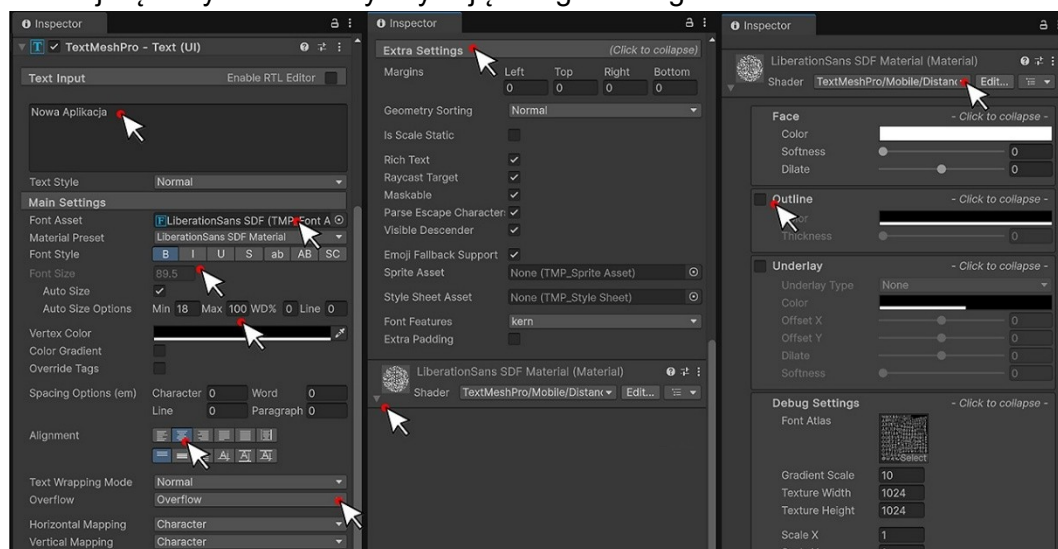
Następnym krokiem jest dodanie tytułu naszej aplikacji. W tym celu dodajemy obiekt typu **Text** - **TextMeshPro**. TextMeshPro jest biblioteką odpowiadającą za zaawansowane wyświetlanie tekstu. Jeżeli w naszym projekcie jej nie ma, to musimy ją pobrać po otwarciu się okna **TMP Importer**. Pobierzmy zarówno **TMP Essentials** jak i **TMP Examples & Extras**.



Rys. 25. Import biblioteki TMP

Następnie umieszczamy tekst w wybranym przez nas miejscu i przechodzimy do edycji jego ustawień. Biblioteka TextMeshPro pozwala na dostosowanie wielu zmiennych związanych z dodanym przez nas napisem w komponencie **TextMeshPro - Text (UI)**. Pierwsze pole określa treść pola testowego. Następnie wybieramy styl tekstu, czcionkę (można pobrać własne czcionki i wrzucić do projektu) oraz jej wielkość. Możemy zaznaczyć również pole **Auto Size**, a wielkość tekstu zostanie automatycznie dopasowana do wymiarów pola tekstowego w granicach określonych w opcjach (wielkość minimalna i maksymalna).

Do obiektu tekstowego przypisany jest również materiał, który znajduje się na samym dole okna **Inspektora**. Materiał określa wygląd czcionki z której korzysta ten komponent. Należy pamiętać, że zmiany wprowadzone wewnątrz komponentu będą dotyczyły tylko tego aktualnego obiektu tekstowego, jednak zmiany wprowadzone w ustawieniach materiału czcionki obejmą wszystkie obiekty używające tego samego materiału.

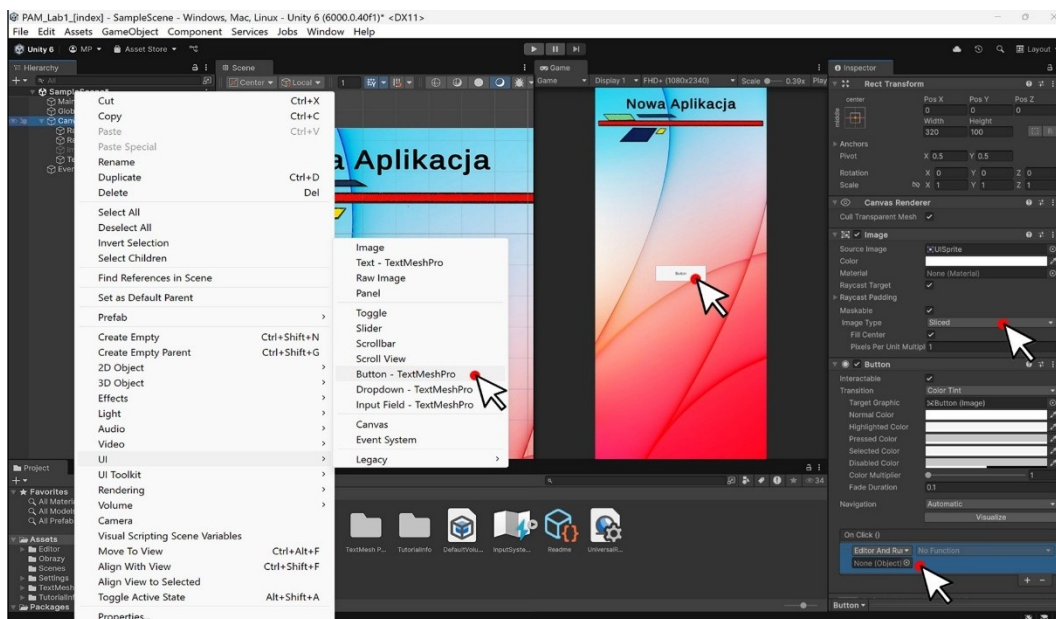


Rys. 26. Ustawienia obiektu tekstowego

- Więcej na ten temat można znaleźć w tym artykule - [TextMesh Pro: Importing the Package](#).

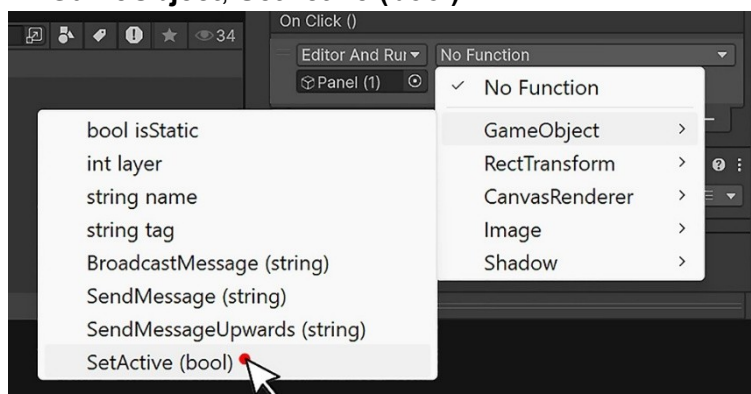
8. Przyciski i interakcje

Kolejnym etapem jest dodanie przycisku do projektowanego ekranu, aby umożliwić użytkownikowi interakcję z aplikacją.



Rys. 27. Ustawienia przycisku

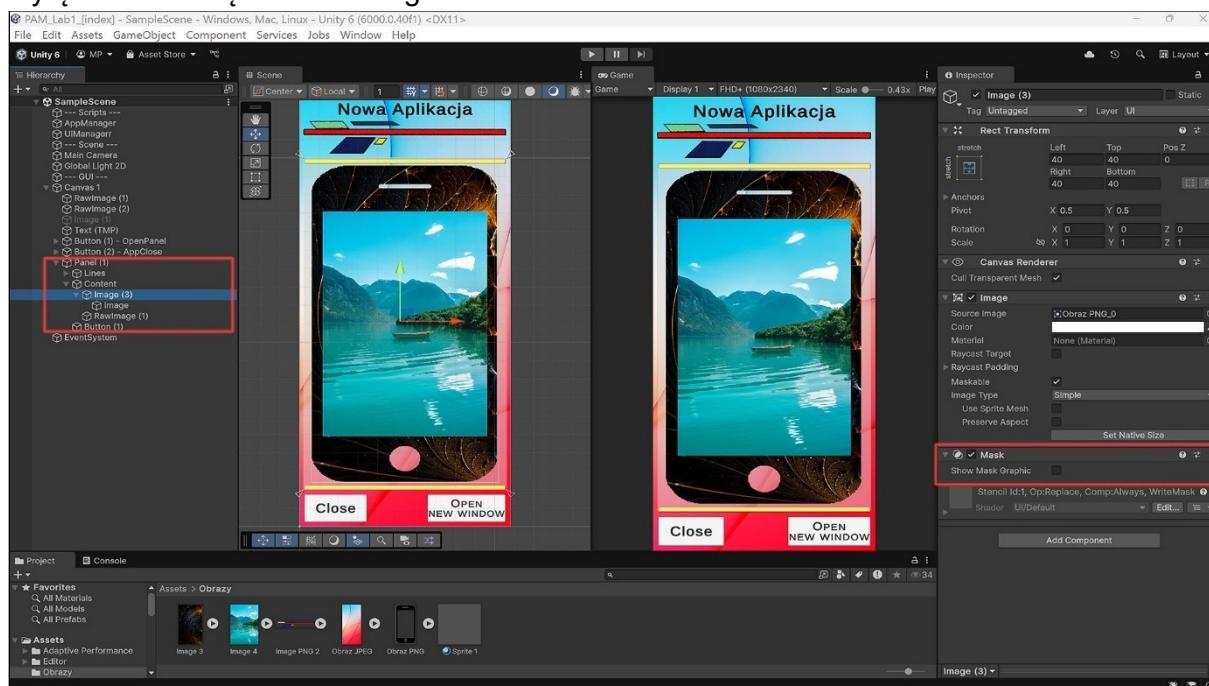
Główne ustawienia dotyczące tego jaką funkcję ma zrealizować naciśnięciu określamy w polu **On Click ()** znajdującego się na samym dole okna Inspektora. Z wybranej listy predefiniowanych funkcji możemy wpłynąć na dodany do tego pola obiekt lub jego komponenty. Dodajmy więc dwa przyciski na dole ekranu. Aby przetestować więc podstawowe funkcje utwórzmy nowy obiekt typu **Panel** wewnątrz którego dodamy kilka obiektów odpowiadających za wyświetlanie dowolnych danych. Pierwszy z naszych przycisków, będzie odpowiadał za zamknięcie aplikacji, a drugi za otwarcie nowoutworzonego panelu. Dlatego obiekt ten - nazwijmy **Panel (1)** - przeciągnijmy do pola przycisku **On Click ()** i wybierzmy działanie z kategorii **GameObject, SetActive (bool)**.



Rys. 28. Wybór funkcji wywołanej przy naciśnięciu przycisku

Ustawiona wartość **bool** zostanie przypisana funkcji **SetActive()** wywołanej a wyznaczonym obiekcie. Dlatego ustawiamy wartość **True**, aby obiekt został aktywowany. Wewnątrz panelu z kolei, utwórzmy przycisk odpowiedzialny za zamykanie go, gdzie analogiczną funkcję przycisk będzie realizował, ale dla wartości **False**. Przycisk ten może być umieszczony klasycznie - gdzieś w obrębie panelu, lub rozciągnięty na cały jego obszar. Wtedy, niezależnie od tego gdzie (wewnątrz panelu) naciśnie użytkownik, to zostanie obiekt ten zamknięty. Jednak aby obraz przycisku nie przysłaniał zawartości, to zmienimy jego przezroczystość na 0. Następnie nasze obiekty możemy uporządkować w odpowiednie kategorie. Ambitni studenci

mogą spróbować użyć komponentu maski dla obrazu. Sprawi to, iż obraz będzie wyświetlał wyłącznie wewnątrz określonego kształtu.



Rys. 29. Przykład użycia maski obrazu

Ostatecznie powinniśmy mieć działającą aplikację pozwalającą na otwieranie i zamykanie dodatkowego panelu z danymi. Aby ją przetestować należy nacisnąć przycisk PLAY, reprezentowany przez strzałkę w górnej części okna edytora Unity.

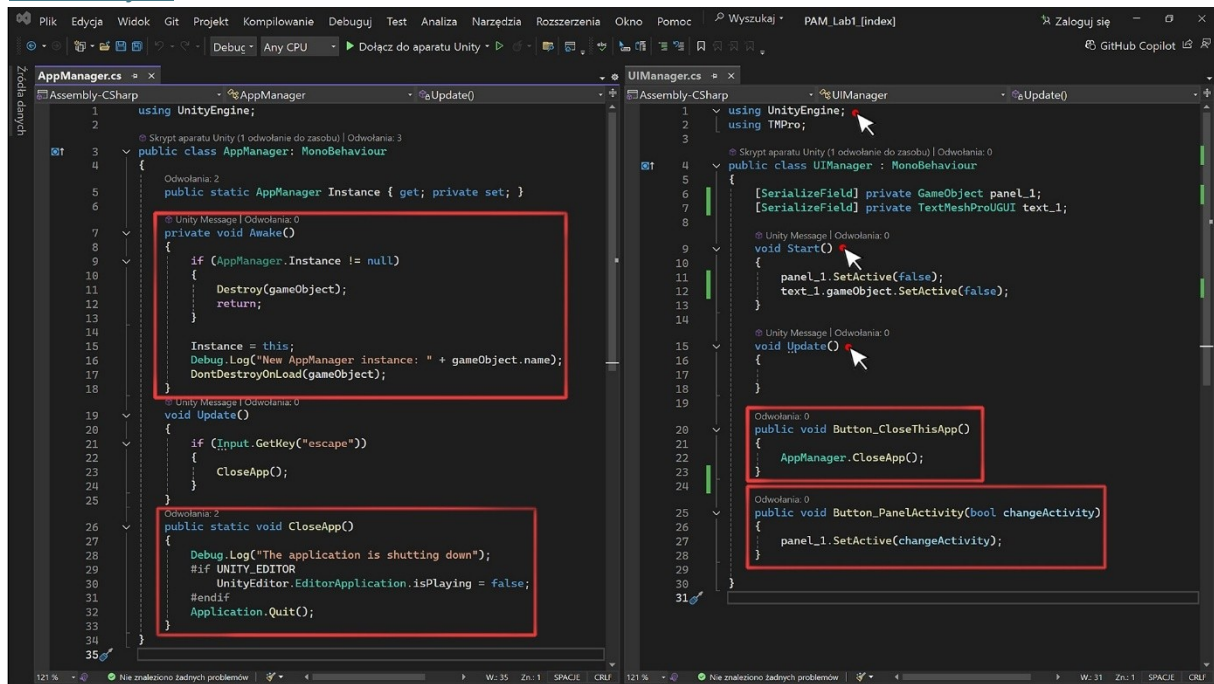
- Więcej na temat interakcji użytkownika z aplikacją mobilną Unity można przeczytać w artykule [Mobile device input](#) lub [Input System package](#).

9. Nowy skrypt C#

Aby móc realizować - poprzez naciśnięcie przycisku - funkcje określone przez samego siebie, konieczne jest wyrażenie ich poprzez kod programu w języku C#. Aby móc to zrealizować, utworzymy nowy folder o nazwie **Scripts**, a w nim obiekt typu **MonoBehaviour Script** i nazwijmy go **AppManager**. Wewnątrz tego skryptu utworzymy klasę, która będzie (przynajmniej powinna być) znanym wzorcem projektowym typu **Singleton**. Wzorec ten oznacza to, iż, że w aplikacji powinna istnieć tylko jedna instancja tej konkretnej klasy. Ułatwia to również każdej innej klasie dostęp do tej instancji za pomocą publicznego statycznego odniesienia. Wzorec ten ma swoje wady i zalety. W niektórych sytuacjach może być on wygodnym rozwiązaniem, podczas gdy w innych powinno się go wręcz unikać.

Utwórzmy również w oknie hierarchii nowy, pusty obiekt poprzez naciśnięcie prawego przycisku myszy, a następnie **Create Empty**. Nazwijmy go również **AppManager**, a następnie przeciągnijmy do niego nasz skrypt jako nowy komponent. Utwórzmy także nowy skrypt o nazwie **UIManager** i również dodajmy go do nowego, pustego obiektu o takiej samej nazwie. Idea tej architektury jest taka, iż wewnątrz skryptu **AppManager** zawarte będą funkcje, do których **UIManager** będzie mógł się odwołać bez posiadania bezpośredniego odniesienia. Samo rozwiązanie jest oczywiście nazbyt skomplikowanym rozwiązaniem obecnego zagadnienia, jednak stanowi sposobność do zaprezentowania Wam powiązań pomiędzy obiektami. Po wykonaniu wszystkich opisanych czynności edytujmy oba skrypty w preferowanym przez nas IDE.

Przykład programowej realizacji został zaprezentowany na poniższym rysunku. Struktura kodu przyjmuje znaną już Wam formę. Na jego początku importowane są biblioteki, następnie definiowana klasa, jej zmienne oraz funkcje (metody). Dodatkowo występuje metoda `Start()` oraz `Update()`, gdzie pierwsza z nich uruchamiana jest przy inicjalizacji skryptu, a druga wywoływana jest w każdej klatce aplikacji. Zwróćcie uwagę na to, w jakich metodach występują zmienne wejściowe lub nie. Po przeciągnięciu obiektu ze skryptem w pole **OnClick()** przycisku wybieramy co i z jaką zmienną wejściową przycisk ten ma realizować. Najczęściej używane typy zmiennych to **string**, **int**, **float**, **double**, [Vector2](#), [Vector3](#) czy [GameObject](#).



Rys. 30. Przykład realizacji programowej

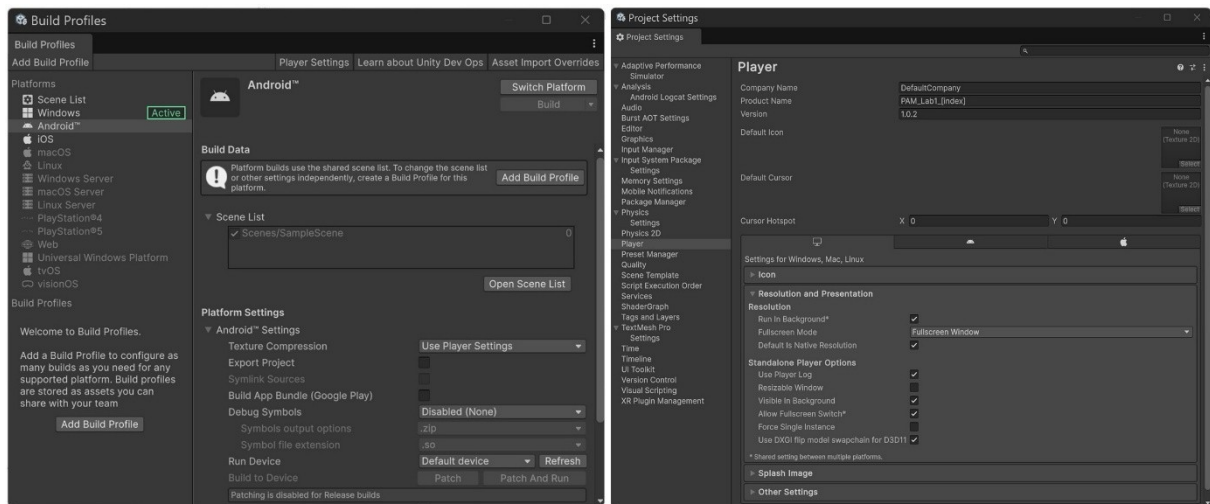
Istnieje jednak bardziej poprawny sposób na zarządzanie cyklem życia aplikacji mobilnej, w tym jej zamykania. Więcej na ten temat można znaleźć w artykule [Quit a Unity Android application](#) lub [How do I programmatically quit my iOS application?](#)

Czy można inaczej zaprojektować taką interakcję? Ciekawym pomysłem może być użycie klasy [Event](#). Przykłady zastosowania: [EventType](#), [Event.Use](#), [Event.clickCount](#). [Touch](#)

- Więcej na temat skryptów można przeczytać w artykule [Get started with scripting](#).
- Więcej na temat zapisywania danych można przeczytać w artykule [PlayerPrefs](#).
- Więcej na temat wzorców projektowych można przeczytać w artykule [Design Patterns in Unity](#).

10. Zadania dla studenta

- 1) Przetestowanie aplikacji za pomocą narzędzia [Profiler](#),
- 2) Zmiana platformy na Android lub iOS,
- 3) Edycja ustawień projektu, dodanie ikony aplikacji,
- 4) Build aplikacji dla określonej platformy,

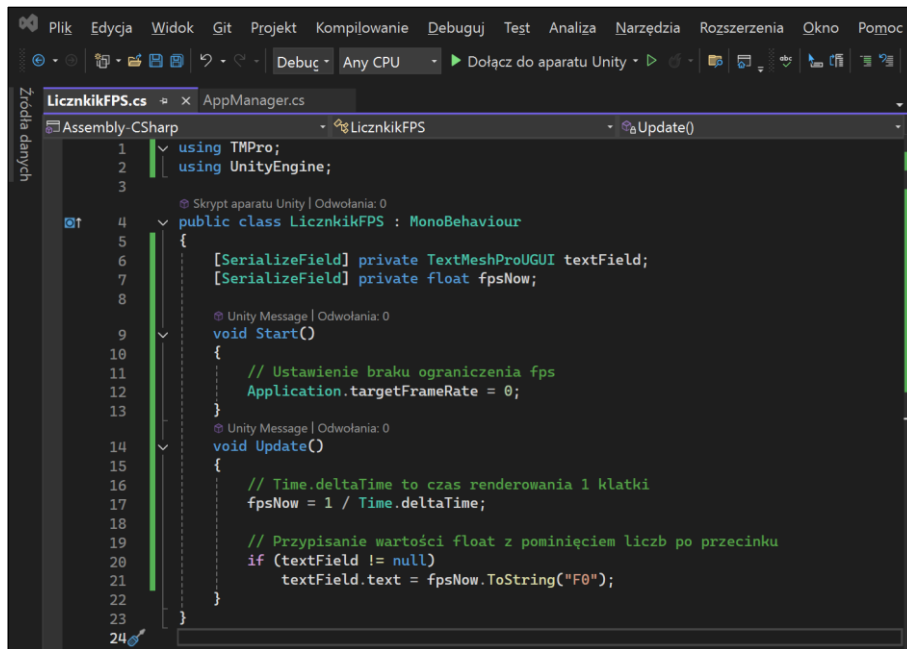


Rys. 31. Ustawienia projektu

- Więcej o narzędziu **Profiler-a** można znaleźć w artykule [UI Profiler](#).
- Więcej o optymalizacji aplikacji mobilnej można znaleźć w artykule [Optimization for Android](#)
- Więcej o dostępie do **IMU** można znaleźć w artykule [Class Accelerometer](#).
- Więcej o ustawieniach użytkownika można przeczytać w artykule [Player settings](#).
- Więcej o publikacji aplikacji można przeczytać w artykule [How to publish to Android](#) oraz [Publishing for iOS](#).

11. Zadania dodatkowe

- 1) Dodanie i obsługa [Slider-a](#),
- 2) Dodanie *Splash screen* (ekranu powitalnego),
- 3) Implementacja skryptu odpowiedzialnego za obliczanie aktualnej ilości klatek na sekundę, przypisanie tej wartości do zmiennej float, a następnie powiązanie jej z polem tekstowym biblioteki TextMeshPro widocznym w oknie aplikacji.



Rys. 32. Częściowa implementacja programowa dla zadania dodatkowego 3

ĆWICZENIE LABORATORYJNE - UNITY 2

Sceny · PlayerPrefs · Animacje · Audio

Programowanie Aplikacji Mobilnych - moduł U

1. Spis treści

1. Spis treści	22
2. Wprowadzenie i przypomnienie z Lab 1	22
3. Zarządzanie scenami (SceneManager)	23
4. Animacje UI - Animator i przejścia	24
5. Korutyny (coroutines)	25
6. Zdarzenia: UnityEvent i delegaty	26
7. Zapis i odczyt danych PlayerPrefs	27
8. Audio w aplikacji mobilnej	28
9. Praktyczne ćwiczenie: nowy skrypt C#	29
10. Zadania dla studenta	31
11. Zadania dodatkowe	31

2. Wprowadzenie i przypomnienie z Lab 1

Niniejsza instrukcja stanowi kontynuację **Laboratorium Unity 1**, w którym zapoznano się ze środowiskiem edytora Unity, tworzeniem projektu 2D Mobile, strukturą edytora, obiektem Canvas oraz podstawowymi elementami UI: obrazami, tekstem i przyciskami. Zostały omówione również skrypty C# i wzorzec **Singleton**.

W niniejszym laboratorium skupimy się na zagadnieniach pozwalających budować bardziej złożone i funkcjonalne aplikacje mobilne:

- zarządzaniu wieloma scenami i nawigacji między nimi,
- tworzeniu animacji UI za pomocą komponentu Animator,
- korutynach jako narzędziu do operacji rozłożonych w czasie,
- zdarzeniach (UnityEvent) i delegatach w C#,
- trwałym zapisie i odczycie danych użytkownika (PlayerPrefs),
- podstawach obsługi dźwięku w aplikacji mobilnej.

Uwaga: Przed przystąpieniem do zajęć należy upewnić się, iż projekt z Laboratorium Unity 1 jest dostępny i otwierany poprawnie w edytorze Unity 6 (6000.0.x).

3. Zarządzanie scenami (SceneManager)

Aplikacje mobilne rzadko składają się z jednego ekranu. Typowa architektura obejmuje przynajmniej: ekran powitalny (Splash Screen), menu główne, właściwy ekran aplikacji oraz ekran ustawień lub podsumowania. Każdy z tych widoków jest w Unity reprezentowany jako osobna scena.

Scena (*ang.* **Scene**) to samodzielna jednostka projektu Unity zawierająca hierarchię obiektów GameObjects oraz skonfigurowane komponenty. Projekt może zawierać dowolną liczbę scen.

2.1. Tworzenie nowej sceny

Nową scenę tworzymy w oknie Project (dolna część edytora) poprzez naciśnięcie prawego przycisku myszy w folderze Scenes i wybór Create > Scene. Alternatywnie możemy użyć menu File > New Scene. Po nadaniu nazwy (np. MenuScene, GameScene) scena pojawia się na liście plików projektu.

2.2. Build Settings - rejestracja scen

Aby Unity mógł wczytywać sceny w czasie działania aplikacji, każda z nich musi zostać dodana do listy **Build Settings**. Otwieramy okno poprzez menu **File > Build Settings** i przeciągamy pliki scen z okna Project do sekcji **Scenes In Build**. Kolejność na liście odpowiada indeksom scen (0, 1, 2...).

Ważne: Scena o indeksie 0 jest uruchamiana jako pierwsza podczas startu aplikacji. Upewnijmy się, że na pierwszej pozycji znajduje się nasz ekran powitalny lub menu główne.

2.3. SceneManager - wczytywanie scen

Do programowego przełączania scen służy klasa *UnityEngine.SceneManagement.SceneManager*. Wczytanie sceny odbywa się za pomocą metody LoadScene(), która przyjmuje jako argument nazwę sceny (string) lub jej indeks (int).

Przykład skryptu MenuManager zarządzającego nawigacją między scenami

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuManager : MonoBehaviour
{
    // Wczytanie sceny po nazwie
    public void LoadGameScene()
    {
        SceneManager.LoadScene("GameScene");
    }

    // Wczytanie sceny po indeksie
    public void LoadByIndex(int index)
    {
        SceneManager.LoadScene(index);
    }

    // Ponowne wczytanie bieżącej sceny
    public void ReloadScene()
    {
        int current = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(current);
    }
}
```

```
}
}
```

2.4. LoadSceneMode, tryby ładowania

Metoda `LoadScene()` może działać w dwóch trybach określanych parametrem **LoadSceneMode**:

1. **Single** (domyślny) - rozładowuje bieżącą scenę i wczytuje nową.
1. **Additive** - wczytuje nową scenę bez usuwania bieżącej, co pozwala na budowanie złożonych środowisk z wielu scen.

Przykład ładowania sceny w trybie Additive

```
// Tryb Additive - obie sceny aktywne jednocześnie
SceneManager.LoadScene("UIOverlay", LoadSceneMode.Additive);

// Wyładowanie sceny dodanej w trybie Additive
SceneManager.UnloadSceneAsync("UIOverlay");
```

Więcej na temat `SceneManager` można przeczytać w artykule [SceneManager](#).

4. Animacje UI - Animator i przejścia

Animacje nadają aplikacji dynamizm i poprawiają odczucia użytkownika (UX). Unity oferuje dwa główne narzędzia do animacji: system `Animation/Animator` oparty na maszynach stanów oraz nowszą bibliotekę `DOTween` (dostępną przez `Asset Store`). W niniejszej instrukcji skupimy się na wbudowanym systemie.

3.1. Komponent Animator

Każdy obiekt który ma być animowany musi posiadać komponent **Animator**. Dodajemy go przez **Add Component > Miscellaneous > Animator** w oknie Inspektora. Komponent ten wymaga przypisania **Animator Controller** - pliku definiującego stany animacji i przejścia między nimi.

3.2. Animator Controller i okno Animator

`Animator Controller` tworzymy w oknie `Project` (`Create > Animator Controller`) i przypisujemy do pola `Controller` komponentu `Animator`. Po dwukrotnym kliknięciu na plik kontrolera otwiera się okno `Animator`, w którym definiujemy:

- stany (`State`) - każdy stan odpowiada jednej animacji (`Animation Clip`),
- przejścia (`Transition`) - strzałki łączące stany, definiujące warunki zmiany stanu,
- parametry - zmienne (`bool`, `int`, `float`, `trigger`) sterujące przejściami ze skryptu.

3.3. Tworzenie klipu animacji

Klipy animacji tworzymy w oknie **Animation** (`Window > Animation > Animation`). Po wybraniu animowanego obiektu naciskamy **Create** i zapisujemy plik **.anim**. Następnie ustawiamy kursornik na wybranej klatce i edytujemy właściwości obiektu (np. `RectTransform.localPosition`, `CanvasGroup.alpha`). Unity automatycznie rejestruje zmiany jako klatki kluczowe (`Keyframes`).

3.4. Sterowanie animacją z poziomu skryptu

Skrypt sterujący animacją panelu UI

```
using UnityEngine;

public class PanelAnimator : MonoBehaviour
{
    private Animator _animator;

    void Start()
    {
        _animator = GetComponent<Animator>();
    }

    // Wywołanie parametru typu Trigger
    public void ShowPanel()
    {
        _animator.SetTrigger("Show");
    }

    // Sterowanie parametrem bool
    public void SetVisibility(bool isVisible)
    {
        _animator.SetBool("IsVisible", isVisible);
    }
}
```

Więcej na temat systemu animacji można przeczytać w artykule [Animation system overview](#).

5. Korutyny (coroutines)

Korutyna to specjalny rodzaj metody w C#, który pozwala na rozłożenie wykonania kodu na wiele klatek aplikacji. Dzięki temu możemy zrealizować efekty takie jak: opóźnione uruchamianie funkcji, stopniowe zanikanie elementów UI (fade in/out), sekwencyjne ładowanie **zasobów** czy animowane liczniki czasu - bez blokowania głównego wątku gry.

4.1. Składnia korutyny

Korutyna zwraca typ **IEnumerator** i zawiera instrukcję **yield return**, która wstrzymuje jej wykonanie na określony czas lub do następnej klatki.

Korutyny realizujące efekt FadeIn i opóźnione wczytanie sceny

```
using System.Collections;
using UnityEngine;

public class FadeController : MonoBehaviour
{
    [SerializeField] private CanvasGroup canvasGroup;

    void Start()
    {
        // Uruchomienie korutyny
        StartCoroutine(FadeIn(1.5f));
    }

    // Korutyna: stopniowe pojawianie się panelu
    IEnumerator FadeIn(float duration)
    {
        float elapsed = 0f;
        canvasGroup.alpha = 0f;

        while (elapsed < duration)
        {
            elapsed += Time.deltaTime;
        }
    }
}
```

```

        canvasGroup.alpha = elapsed / duration;
        yield return null; // Czekaj do następnej klatki
    }

    canvasGroup.alpha = 1f;
}

// Korutyna: opóźnione przejście do innej sceny
IEnumerator LoadSceneAfterDelay(string sceneName, float delay)
{
    yield return new WaitForSeconds(delay);
    UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName);
}
}

```

Ważne: Komponent **CanvasGroup** pozwala na jednoczesne kontrolowanie przezroczystości (alpha), interaktywności i blokowania raycasting całej grupy elementów UI. Jest to zalecany sposób realizacji efektu fade in/fade out. Więcej na temat korutyn można przeczytać w artykule [Coroutines](#).

6. Zdarzenia: UnityEvent i delegaty

Zdarzenia to mechanizm umożliwiający luźne powiązanie komponentów. Zamiast bezpośredniego wywoływania metod jednego skryptu przez inny, komponent emituje zdarzenie, a subskrybenci reagują niezależnie. Poprawia to modularność i testowalność kodu.

5.1. UnityEvent - zdarzenia konfigurowane w Inspektorze

Klasa **UnityEvent** z przestrzeni nazw **UnityEngine.Events** pozwala na definiowanie zdarzeń, do których metody subskrybujące można przypisać zarówno z poziomu Inspektora (podobnie jak **OnClick()**), jak i z poziomu skryptu.

Przykład zastosowania UnityEvent z parametrem

```

using UnityEngine;
using UnityEngine.Events;

public class HealthSystem : MonoBehaviour
{
    // Zdarzenie wywoływane przy zmianie zdrowia
    public UnityEvent<int> OnHealthChanged;
    // Zdarzenie wywoływane przy śmierci
    public UnityEvent OnDeath;

    private int _health = 100;

    public void TakeDamage(int damage)
    {
        _health -= damage;
        OnHealthChanged?.Invoke(_health);

        if (_health <= 0)
        {
            OnDeath?.Invoke();
        }
    }
}

```

5.2. Delegaty i zdarzenia C# (event/Action)

Alternatywą dla UnityEvent są wbudowane w C# mechanizmy: **delegate**, **event** oraz **Action/Func**. Są one szybsze od UnityEvent, jednak nie są serializowane przez Unity i nie pojawiają się w Inspektorze.

Globalny system zdarzeń z użyciem Action

```
using System;
using UnityEngine;

public class GameEvents : MonoBehaviour
{
    // Statyczne zdarzenie dostępne globalnie
    public static event Action<int> OnScoreChanged;
    public static event Action OnGameOver;

    public static void ScoreChanged(int newScore)
    {
        OnScoreChanged?.Invoke(newScore);
    }

    public static void GameOver()
    {
        OnGameOver?.Invoke();
    }
}

// Subskrybent
public class ScoreUI : MonoBehaviour
{
    void OnEnable() => GameEvents.OnScoreChanged += UpdateScoreText;
    void OnDisable() => GameEvents.OnScoreChanged -= UpdateScoreText;

    void UpdateScoreText(int score)
    {
        // Aktualizacja tekstu TMP...
    }
}
```

Uwaga: Zawsze należy odsubskrybować zdarzenia (operator -=) w metodzie **OnDisable()** lub **OnDestroy()**, aby zapobiec wyciekowi pamięci. Więcej na temat UnityEvent można przeczytać w artykule [UnityEvent](#).

7. Zapis i odczyt danych PlayerPrefs

Aplikacje mobilne często wymagają trwałego przechowywania danych pomiędzy sesjami użytkownika: ustawień głośności, nazwy gracza, najlepszego wyniku czy ukończonego poziomu. Najprostszym rozwiązaniem dostępnym w Unity jest klasa PlayerPrefs.

PlayerPrefs przechowuje dane w postaci par klucz-wartość. Obsługuje trzy typy: **int**, **float** oraz **string**. Dane są zapisywane lokalnie na urządzeniu (np. w rejestrze systemu Windows lub pliku plist na iOS/Android).

6.1. Zapis i odczyt wartości

Operacje na PlayerPrefs, zapis, odczyt, sprawdzenie i usuwanie

```
// Zapis
PlayerPrefs.SetInt("HighScore", 1500);
PlayerPrefs.SetFloat("Volume", 0.75f);
PlayerPrefs.SetString("PlayerName", "Jan");
PlayerPrefs.Save(); // Wymusza natychmiastowy zapis na dysk

// Odczyt (drugi argument to wartość domyślna)
int score = PlayerPrefs.GetInt("HighScore", 0);
float vol = PlayerPrefs.GetFloat("Volume", 1.0f);
string name = PlayerPrefs.GetString("PlayerName", "Gracz");

// Sprawdzenie istnienia klucza
if (PlayerPrefs.HasKey("HighScore"))
```

```
{
    Debug.Log("Klucz istnieje!");
}

// Usunięcie klucza / wszystkich danych
PlayerPrefs.DeleteKey("HighScore");
PlayerPrefs.DeleteAll();
```

6.2. Przykład praktyczny menedżera ustawień

Dobłą praktyką jest enkapsulacja operacji PlayerPrefs w dedykowanej klasie zarządzającej, co ułatwia zmianę systemu persystencji w przyszłości (np. na plik JSON lub bazę danych SQLite).

```
Statyczna klasa SettingsManager jako warstwa abstrakcji nad PlayerPrefs
using UnityEngine;

public static class SettingsManager
{
    private const string KEY_VOLUME = "MasterVolume";
    private const string KEY_NAME = "PlayerName";

    public static float MasterVolume
    {
        get => PlayerPrefs.GetFloat(KEY_VOLUME, 1f);
        set { PlayerPrefs.SetFloat(KEY_VOLUME, value); PlayerPrefs.Save(); }
    }

    public static string PlayerName
    {
        get => PlayerPrefs.GetString(KEY_NAME, "Gracz");
        set { PlayerPrefs.SetString(KEY_NAME, value); PlayerPrefs.Save(); }
    }
}
```

Ograniczenia: PlayerPrefs nie jest bezpiecznym mechanizmem. Dane mogą być łatwo odczytane i zmodyfikowane przez użytkownika. Wrażliwych informacji (tokenów, haseł) nie należy przechowywać w PlayerPrefs. Więcej na temat PlayerPrefs można przeczytać w artykule [PlayerPrefs](#).

8. Audio w aplikacji mobilnej

Dźwięk jest istotnym elementem odczuć użytkownika. Unity oferuje rozbudowany system audio oparty na komponentach AudioSource i AudioListener oraz trójwymiarowej przestrzeni dźwięku.

7.1. Komponenty audio

AudioListener - odpowiednik ucha w scenie. Domyślnie przypisany do obiektu Main Camera. W scenie powinien znajdować się dokładnie jeden taki komponent.

AudioSource - komponent emitujący dźwięk. Może odtwarzać klip jednorazowo lub w pętli, z regulowaną głośnością i efektem przestrzennym.

7.2. Import pliku dźwiękowego

Obsługiwane formaty plików: **.wav**, **.mp3**, **.ogg**. Po przeciągnięciu pliku do okna Project Unity tworzy zasób **AudioClip**. Zalecane ustawienia dla dźwięków UI (przycisków) to **Load Type**:

Decompress On Load (krótkie klipy). Dla muzyki tła stosujemy **Streaming** aby oszczędzić pamięć RAM.

7.3. Odtwarzanie dźwięku ze skryptu

Klasa AudioManager jako Singleton trwający między scenami (DontDestroyOnLoad)

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    public static AudioManager Instance { get; private set; }

    [SerializeField] private AudioSource sfxSource;
    [SerializeField] private AudioSource musicSource;

    void Awake()
    {
        if (Instance != null) { Destroy(gameObject); return; }
        Instance = this;
        DontDestroyOnLoad(gameObject); // Trwa między scenami
    }

    // Odtworzenie dźwięku UI (jednorazowo)
    public void PlaySFX(AudioClip clip)
    {
        sfxSource.PlayOneShot(clip);
    }

    // Zmiana głośności muzyki
    public void SetMusicVolume(float volume)
    {
        musicSource.volume = volume;
        SettingsManager.MasterVolume = volume;
    }
}
```

DontDestroyOnLoad(gameObject) powoduje, że obiekt GameManager nie zostaje zniszczony przy wczytaniu nowej sceny. Jest to standardowa technika dla menedżerów globalnych (audio, gra, ustawienia). Więcej na temat systemu audio można przeczytać w artykule [Audio overview](#).

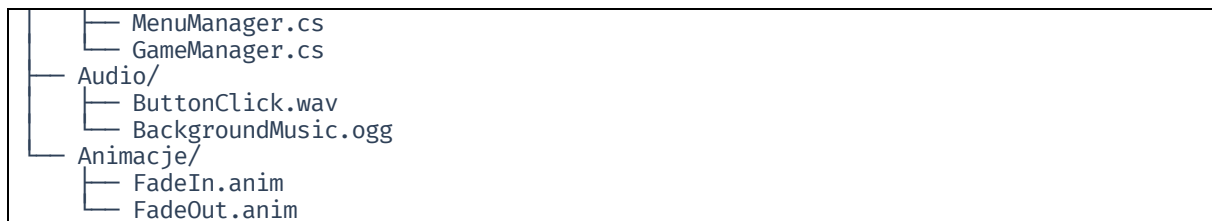
9. Praktyczne ćwiczenie: nowy skrypt C#

Na podstawie wiedzy z niniejszego laboratorium zbudujemy kompletną mini-aplikację składającą się z dwóch scen: ekranu głównego (MenuScene) oraz ekranu gry (GameScene). Aplikacja będzie pamiętać imię gracza, odtwarzać dźwięk naciśnięcia przycisku i płynnie przechodzić między scenami dzięki animacji FadeIn/FadeOut.

8.1. Struktura projektu

Proponowane struktura folderów projektu

```
Assets/
├── Scenes/
│   ├── MenuScene.unity
│   └── GameScene.unity
└── Scripts/
    ├── AudioManager.cs
    ├── SettingsManager.cs
    └── FadeController.cs
```



8.2. Przykład pełnego skryptu MenuManager

Kompletny skrypt MenuManager łączący wszystkie omawiane zagadnienia

```
using UnityEngine;
using UnityEngine.UI;
using TMPPro;
using System.Collections;
using UnityEngine.SceneManagement;

public class MenuManager : MonoBehaviour {
    [Header("UI References")]
    [SerializeField] private TMP_InputField nameInputField;
    [SerializeField] private TextMeshProUGUI welcomeText;

    [Header("Audio")]
    [SerializeField] private AudioClip buttonClickClip;

    [Header("Animacja")]
    [SerializeField] private CanvasGroup fadeCanvasGroup;
    [SerializeField] private float fadeDuration = 0.8f;

    void Start() {
        // Wczytanie zapisanego imienia
        string savedName = SettingsManager.PlayerName;
        welcomeText.text = $"Witaj, {savedName}!";
        nameInputField.text = savedName;

        // Animacja wejścia
        StartCoroutine(FadeIn());
    }

    public void OnStartButtonClicked() {
        // Zapis imienia
        SettingsManager.PlayerName = nameInputField.text;

        // Dźwięk przycisku
        AudioManager.Instance?.PlaySFX(buttonClickClip);

        // Przejście do sceny gry z FadeOut
        StartCoroutine(FadeAndLoad("GameScene"));
    }

    IEnumerator FadeIn() {
        float t = 0;
        while (t < fadeDuration)
        {
            t += Time.deltaTime;
            fadeCanvasGroup.alpha = 1f - (t / fadeDuration);
            yield return null;
        }
        fadeCanvasGroup.alpha = 0f;
    }

    IEnumerator FadeAndLoad(string sceneName) {
        float t = 0;
        while (t < fadeDuration) {
            t += Time.deltaTime;
            fadeCanvasGroup.alpha = t / fadeDuration;
            yield return null;
        }
        SceneManager.LoadScene(sceneName);
    }
}
```

10. Zadania dla studenta

Poniższe zadania należy wykonać samodzielnie, korzystając z wiedzy zdobytej w Laboratorium Unity 1 i 2. Efektem każdego zadania powinno być działające rozwiązanie przetestowane przyciskiem PLAY w edytorze Unity.

1. Stworzenie dwóch scen (MenuScene i GameScene) oraz zaimplementowanie nawigacji między nimi za pomocą skryptu SceneManager. W MenuScene dodaj przycisk Start otwierający GameScene, a w GameScene przycisk Wróć ładujący MenuScene.
2. Dodanie animacji **wejścia** dla panelu UI (FadeIn z efektem przezroczystości CanvasGroup.alpha) uruchamianej automatycznie przy starcie sceny. Czas animacji powinien być konfigurowalny w Inspektorze.
3. Zaimplementowanie systemu zapisu i odczytu imienia gracza z użyciem PlayerPrefs. Imie wpisane w InputField na ekranie menu powinno być wyświetlane jako powitanie w GameScene.
4. Dodanie dźwięku przycisku (AudioSource.PlayOneShot) w odpowiedzi na każde naciśnięcie przycisku w aplikacji. Klip AudioClip przypisać przez Inspektor.
5. Implementacja korutyny odpowiedzialnej za odliczanie czasu (licznik sekundy) z wyświetlaniem wartości w polu TextMeshPro. Czas powinien zatrzymywać się po naciśnięciu przycisku Pauza.

11. Zadania dodatkowe

- Zaimplementowanie globalnego systemu zdarzeń (static event Action) informującego o zmianie wyniku. ScoreUI powinien subskrybować zdarzenie i aktualizować wyświetlany wynik bez bezpośredniego odwołania do GameManager.
- Stworzenie ekranu ustawień (SettingsScene lub panelu w MenuScene) z suwakiem (Slider) regulującym głośność muzyki. Wartość suwaka powinna być zapisywana przez SettingsManager i odtwarzana przy ponownym uruchomieniu aplikacji.
- Implementacja klasy AudioManager jako Singleton z DontDestroyOnLoad, zdolnego do odtwarzania muzyki tła oraz dźwięków UI z osobnych AudioSource. Muzyka powinna nie restartować się przy wczytaniu kolejnej sceny.
- Dodanie ekranu ładowania (Loading Screen) z paskiem postępu (Slider) wyświetlanym podczas asynchronicznego wczytywania sceny (SceneManager.LoadSceneAsync). Pasek powinien odzwierciedlać rzeczywisty postęp operacji (AsyncOperation.progress).

Studenci realizujący zadania dodatkowe mogą skorzystać z dokumentacji Unity Learn:

- Więcej na temat LoadSceneAsync - [LoadSceneAsync](#).
- Więcej na temat AsyncOperation.progress - [AsyncOperation.progress](#).
- Więcej na temat DontDestroyOnLoad - [DontDestroyOnLoad](#).
- Więcej na temat Audio Mixer - [Audio Mixer](#).

ĆWICZENIE LABORATORYJNE - UNITY 3

Sceny · PlayerPrefs · Animacje · Audio

Programowanie Aplikacji Mobilnych - moduł U

1. Spis treści

1. Spis treści	32
2. Wprowadzenie i przegląd zagadnień mobilnych	32
3. Wejście dotykowe: Touch Input	33
4. Gesty użytkownika - Swipe, Pinch, Tap	35
5. Czujniki urządzenia: akcelerometr i żyroskop	37
6. Nowy Input System	39
7. Bezpieczny obszar ekranu - Safe Area i Notch	40
8. Orientacja ekranu i responsywność UI	42
9. Optymalizacja wydajności na platformach mobilnych	43
10. Budowanie i wdrożenie aplikacji (Build & Deploy)	45
11. Zadania dla studenta	47
12. Zadania dodatkowe	48

2. Wprowadzenie i przegląd zagadnień mobilnych

Poprzednie laboratoria (Unity 1 i Unity 2) skupiały się na podstawach środowiska Unity, budowaniu interfejsu użytkownika, nawigacji między scenami, animacjach, korutynach, zdarzeniach oraz trwałym zapisie danych. Niniejsza instrukcja przenosi te umiejętności na grunt prawdziwego **urządzenia** mobilnego, omawiając zagadnienia specyficzne wyłącznie dla platform Android i iOS.

Kluczowa różnica między aplikacją desktopową, a mobilną polega na sposobie **wejścia** użytkownika. Zamiast myszy i klawiatury pojawia się dotykowy ekran wielopunktowy, a zamiast joysticka - wbudowane czujniki inercyjne. Dodatkową specyfiką platform mobilnych jest ogromne zróżnicowanie rozdzielczości i proporcji ekranów, obecność wcięcia (notch) lub otworu (punch-hole) w ekranie oraz surowe ograniczenia termiczne i energetyczne.

Po wykonaniu czynności opisanych w tej instrukcji student będzie potrafił:

- obsługiwać wejście dotykowe jednym i wieloma palcami,
- rozpoznawać typowe gesty: tap, swipe, pinch-to-zoom,
- odczytywać dane z akcelerometru i żyroskopu,
- korzystać z nowego Input System dla platform mobilnych,
- dostosowywać UI do bezpiecznego obszaru ekranu (Safe Area),
- zarządzać orientacją ekranu programowo i przez ustawienia projektu,
- profilować i optymalizować aplikacje pod kątem zużycia CPU/GPU/RAM,
- budować i wdrażać plik APK/AAB na urządzenie z systemem Android.

3. Wejście dotykowe: Touch Input

Ekran dotykowy jest głównym interfejsem **wejścia urządzenia** mobilnego. Unity udostępnia dwa sposoby odczytu dotyku: starszy, oparty na klasie Input, oraz nowszy, oparty na pakiecie Input System. W tej sekcji omówimy obie metody, zaczynając od klasycznego API.

2.1. Klasa Input: Touch i TouchPhase

Każdy kontakt palca z ekranem reprezentowany jest przez strukturę **Touch**. Dostępna jest przez właściwość **Input.touches** (tablica aktywnych dotknięcia) lub metodę **Input.GetTouch(int index)**.

Pole struktury Touch	Opis
fingerId	Unikalny identyfikator palca (przydatny przy multitouch)
position	Pozycja dotknięcia na ekranie w pikselach (Vector2)
deltaPosition	Zmiana pozycji od ostatniej klatki (Vector2)
phase	Faza dotknięcia: Began, Moved, Stationary, Ended, Canceled
tapCount	Liczba stuknieć (single tap, double tap...)
pressure	Sila nacisku (0..1), jeśli urządzenie to obsługuje

Podstawowa obsługa dotyku przez klasę Input

```
using UnityEngine;

public class TouchInputBasic : MonoBehaviour
{
    void Update()
    {
        // Iteracja po wszystkich aktywnych dotknięciach
        for (int i = 0; i < Input.touchCount; i++)
        {
            Touch touch = Input.GetTouch(i);

            switch (touch.phase)
            {
                case TouchPhase.Began:
                    Debug.Log($"Palec {touch.fingerId} dotknął ekranu @ {touch.position}");
                    break;

                case TouchPhase.Moved:
                    Debug.Log($"Palec {touch.fingerId} przesunął się o {touch.deltaPosition}");
                    break;

                case TouchPhase.Ended:
```

```

        Debug.Log($"Palec {touch.fingerId} opuszczony");
        break;
    }
}
}

```

2.2. Raycast z dotyku - wykrywanie trafionego obiektu UI

Częstym zadaniem jest sprawdzenie, który element UI lub obiekt sceny został dotknięty. Dla obiektów UI służy do tego GraphicRaycaster i EventSystem, natomiast dla obiektów 3D/2D w przestrzeni świata - standardowy Physics2D.Raycast lub Camera.ScreenPointToRay.

Raycast z punktu dotyku, wykrywanie trafionego obiektu 2D

```

using UnityEngine;
using UnityEngine.EventSystems;
using System.Collections.Generic;

public class TouchRaycast : MonoBehaviour
{
    [SerializeField] private Camera mainCamera;

    void Update()
    {
        if (Input.touchCount == 0) return;
        Touch touch = Input.GetTouch(0);
        if (touch.phase != TouchPhase.Began) return;

        // Sprawdzenie czy dotyk trafil w element UI
        if (EventSystem.current.IsPointerOverGameObject(touch.fingerId))
        {
            Debug.Log("Dotyk trafil w element UI - pomijam raycast 3D");
            return;
        }

        // Raycast w przestrzeni 2D
        Vector2 worldPos = mainCamera.ScreenToWorldPoint(touch.position);
        RaycastHit2D hit = Physics2D.Raycast(worldPos, Vector2.zero);

        if (hit.collider != null)
        {
            Debug.Log($"Trafiono obiekt: {hit.collider.gameObject.name}");
        }
    }
}

```

WSKAZÓWKA

Metoda `EventSystem.current.IsPointerOverGameObject(fingerId)` jest kluczowa przy łączeniu UI z przestrzenią gry. Bez tego sprawdzenia kliknięcie przycisku jednocześnie wyzwoi zdarzenia UI i raycast 3D, prowadząc do nieoczekiwanych zachowań.

2.3. Symulacja dotyku w edytorze

Testowanie wejścia dotykowego bez fizycznego **urządzenia** jest możliwe dzięki opcji Simulate Touch Input w Unity. Włączamy ją przez menu Edit > Project Settings > Input Manager lub przez Inspector urządzenia w oknie Device Simulator (Window > General > Device Simulator).

WSKAZÓWKA

Zalecany przepływ pracy: szybkie iteracje w edytorze z Device Simulator, a pełne testy na fizycznym urządzeniu przed oddaniem projektu. **Niektórych** efektów (np. siły nacisku, wibacji) nie da się zasymulować w edytorze.

- Dokumentacja Unity: [Input.Touch](#)
- Dokumentacja Unity: [Device Simulator](#)

4. Gesty użytkownika: Swipe, Pinch, Tap

Gesty są abstrakcją wyższego rzędu zbudowana na podstawie surowych danych dotyku. Unity nie dostarcza gotowych dekodów gestów w warstwie podstawowej - konieczna jest ich implementacja lub użycie gotowej biblioteki (np. Lean Touch z Asset Store). Poniżej omówimy ręcznie implementowane gesty, co daje pełną kontrolę nad zachowaniem aplikacji.

3.1. Tap, stuknięcie

Tap to krótki dotyk bez istotnego przesunięcia. Wykrywamy go porównując czas trwania i deltę pozycji dotknięcia.

Detektor gestu Tap oparty na czasie trwania i długości przesunięcia

```
using UnityEngine;

public class TapDetector : MonoBehaviour
{
    private const float MAX_TAP_DURATION = 0.3f; // sekundy
    private const float MAX_TAP_MOVE_SQ = 400f; // piksele^2

    private float _touchStartTime;
    private Vector2 _touchStartPos;

    void Update()
    {
        if (Input.touchCount != 1) return;
        Touch touch = Input.GetTouch(0);

        if (touch.phase == TouchPhase.Began)
        {
            _touchStartTime = Time.time;
            _touchStartPos = touch.position;
        }
        else if (touch.phase == TouchPhase.Ended)
        {
            float duration = Time.time - _touchStartTime;
            float moveSq = (touch.position - _touchStartPos).sqrMagnitude;

            if (duration < MAX_TAP_DURATION && moveSq < MAX_TAP_MOVE_SQ)
            {
                OnTap(touch.position);
            }
        }
    }

    private void OnTap(Vector2 screenPos)
    {
        Debug.Log($"TAP @ {screenPos}");
    }
}
```

3.2. Swipe, przeciągnięcie

Swipe to szybkie przeciągnięcie palca w określonym kierunku. Po uniesieniu palca obliczamy wektor ruchu i na jego podstawie określamy kierunek (lewo, prawo, góra, dół).

Detektor gestu Swipe z czterema kierunkami i zdarzeniami UnityEvent

```
using UnityEngine;
using UnityEngine.Events;

public class SwipeDetector : MonoBehaviour
{
    [SerializeField] private float minSwipeDistance = 100f; // px
    [SerializeField] private float maxSwipeDuration = 0.5f; // s

    public UnityEvent OnSwipeLeft;
    public UnityEvent OnSwipeRight;
    public UnityEvent OnSwipeUp;
    public UnityEvent OnSwipeDown;

    private Vector2 _startPos;
    private float _startTime;

    void Update()
    {
        if (Input.touchCount != 1) return;
        Touch touch = Input.GetTouch(0);

        if (touch.phase == TouchPhase.Began)
        {
            _startPos = touch.position;
            _startTime = Time.time;
        }
        else if (touch.phase == TouchPhase.Ended)
        {
            float duration = Time.time - _startTime;
            if (duration > maxSwipeDuration) return;

            Vector2 delta = touch.position - _startPos;
            if (delta.magnitude < minSwipeDistance) return;

            // Dominujący kierunek
            if (Mathf.Abs(delta.x) > Mathf.Abs(delta.y))
                (delta.x > 0 ? OnSwipeRight : OnSwipeLeft).Invoke();
            else
                (delta.y > 0 ? OnSwipeUp : OnSwipeDown).Invoke();
        }
    }
}
```

3.3. Pinch-to-Zoom; szczypanie

Pinch to gest z użyciem dwóch palców służący do powiększania i pomniejszania widoku. Mierzmy zmianę odległości między palcami i stosujemy ją do skali obiektu lub orthographicSize kamery.

Implementacja Pinch-to-Zoom dla kamery ortograficznej

```
using UnityEngine;

public class PinchZoom : MonoBehaviour
{
    [SerializeField] private Camera cam;
    [SerializeField] private float minOrtho = 2f;
    [SerializeField] private float maxOrtho = 10f;
    [SerializeField] private float zoomSpeed = 0.01f;

    void Update()
    {
        // Implementacja gestu Pinch-to-Zoom
    }
}
```

```

        if (Input.touchCount != 2) return;

        Touch t0 = Input.GetTouch(0);
        Touch t1 = Input.GetTouch(1);

        // Pozycje palcow w poprzedniej klatce
        Vector2 t0prev = t0.position - t0.deltaPosition;
        Vector2 t1prev = t1.position - t1.deltaPosition;

        float prevDist = Vector2.Distance(t0prev, t1prev);
        float currDist = Vector2.Distance(t0.position, t1.position);
        float delta     = prevDist - currDist; // ujemne = rozsuniecie = zoom in

        cam.orthographicSize = Mathf.Clamp(
            cam.orthographicSize + delta * zoomSpeed,
            minOrtho,
            maxOrtho
        );
    }
}

```

- Dokumentacja Unity: [Input.touches](#)
- Dokumentacja Unity: [Lean Touch \(Asset Store\)](#)

5. Czujniki urządzenia: akcelerometr i żyroskop

Współczesne smartfony i tablety zawierają zestaw czujników inercyjnych: akcelerometr, żyroskop, magnetometr (kompas) i barometr. Najczęściej używanymi w aplikacjach mobilnych są akcelerometr i żyroskop.

4.1. Akcelerometr - Input.acceleration

Akcelerometr mierzy przyspieszenie liniowe **urządzenia** w trzech osiach, włącznie z przyspieszeniem grawitacyjnym. Zwracana wartość to **Vector3** w jednostki g (9.81 m/s²). Przy urządzeniu trzymanym pionowo w pionie: **x ≈ 0**, **y ≈ -1**, **z ≈ 0**.

Os	Znaczenie (urządzenie pionowo)
X (acceleration.x)	Przechył lewo/prawo (landscape tilt)
Y (acceleration.y)	Przechył przód/tył (pitch) + grawitacja (-1 w pionie)
Z (acceleration.z)	Obrot wzdłuż osi przód-tył (roll)

Sterowanie obiektem akcelerometrem z filtrem dolnoprzepustowym

```

using UnityEngine;

public class AccelerometerDemo : MonoBehaviour
{
    [SerializeField] private float sensitivity = 3f;
    [SerializeField] private float maxSpeed    = 5f;

    // Filtrowanie dolnoprzepustowe - wygładza drżenie czujnika
    private Vector3 _filteredAccel;
    private const float LOW_PASS = 0.1f;

    void Update()
    {
        // 1. Odczyt surowy
        Vector3 rawAccel = Input.acceleration;

        // 2. Wygładzanie (low-pass filter)
        _filteredAccel = Vector3.Lerp(_filteredAccel, rawAccel, LOW_PASS);
    }
}

```

```
// 3. Przeszczepienie obiektu w osi X i Y
float moveX = _filteredAccel.x * sensitivity;
float moveY = _filteredAccel.y * sensitivity;

Vector3 move = new Vector3(moveX, moveY, 0f);
move = Vector3.ClampMagnitude(move, maxSpeed);

transform.position += move * Time.deltaTime;
}
}
```

UWAGA

Surowe odczyty akcelerometru zawierają szumy i drżenie. Zawsze należy stosować filtr dolnoprzepustowy (low-pass filter) realizowany przez `Vector3.Lerp` lub dedykowaną klasę `SensorUpdateInterval`, by uniknąć trzesacego się ruchu obiektu.

4.2. Żyroskop; Input.gyro

Żyroskop mierzy prędkość kątów obrotu **urządzenia**. W Unity dostępny jest przez **Input.gyro**. Przed użyciem należy go włączyć, gdyż domyślnie jest wyłączony w celu oszczędzania energii.

Obrót obiektu zgodny z orientacją **urządzenia** na podstawie żyroskopu

```
using UnityEngine;

public class GyroscopeDemo : MonoBehaviour
{
    private bool _gyroSupported;

    // Korekcja orientacji między układem współrzędnych Unity a czujnika
    private static readonly Quaternion baseRotation =
        Quaternion.Euler(90, 0, 0);

    void Start()
    {
        _gyroSupported = SystemInfo.supportsGyroscope;
        if (_gyroSupported)
        {
            Input.gyro.enabled = true;
            Input.gyro.updateInterval = 0.0167f; // ~60 Hz
        }
        else
        {
            Debug.LogWarning("Urządzenie nie posiada żyroskopu!");
        }
    }

    void Update()
    {
        if (!_gyroSupported) return;

        // Konwersja rotacji z układu czujnika do układu Unity
        Quaternion gyroRot = Input.gyro.attitude;
        Quaternion unityRot = new Quaternion(gyroRot.x, gyroRot.y, -gyroRot.z, -gyroRot.w);
        transform.rotation = baseRotation * unityRot;
    }
}
```

4.3. Sprawdzanie dostępności czujników

Nie każde urządzenie mobilne posiada wszystkie czujniki. Przed użyciem należy sprawdzić ich dostępność przez klasę `SystemInfo`:

Sprawdzanie dostępności czujników i użycie wibracji

```
// Sprawdzenie dostępności czujników
bool hasGyro      = SystemInfo.supportsGyroscope;
bool hasAccel     = SystemInfo.supportsAccelerometer;
bool hasVibration = SystemInfo.supportsVibration;

// Wibracja (haptyczne sprzężenie zwrotne)
if (hasVibration)
    Handheld.Vibrate();
```

- Dokumentacja Unity: [Input.gyro](#)
- Dokumentacja Unity: [Input.acceleration](#)
- Dokumentacja Unity: [SystemInfo](#)

6. Nowy Input System

Od Unity 2019.1 dostępny jest nowy pakiet Input System, będący następcą starego Unity Input Manager. Oferuje ujednolicone API dla wszystkich platform, włącznie z dotykiem, żyroskopem/akcelerometrem, gamepadami i klawiaturą. Szczególnie przydatny jest przy budowaniu wieloplatformowych gier, gdyż pozwala łączyć różne źródła **wejścia** w jedno logiczne działanie (Action).

5.1. Instalacja pakietu

Input System instalujemy przez **Window > Package Manager** wyszukując **Input System** i klikając Install. Po instalacji Unity pyta o zmianę Active Input Handling: wybieramy **Both** (obsługa starego i nowego API jednocześnie) lub **New Input System Backend** (tylko nowe API). Edytor wymaga restartu.

UWAGA

Włączenie trybu New Input System Backend wyłącza klasę Input (stare API). Jeżeli projekt korzysta z bibliotek używających starego API (np. starsze wersje Cinemachine), należy wybrać tryb Both.

5.2. Input Actions Asset

Centralna koncepcja nowego systemu jest plik Input Actions (rozszerzenie .inputactions). Tworzymy go przez Create > Input Actions w oknie Project. Plik definiuje:

- Action Map - logiczna grupa akcji (np. Player, UI, Gameplay),
- Action - konkretna czynność użytkownika (np. Move, Jump, Touch),
- Binding - mapowanie fizyczne (np. <Touchscreen>/primaryTouch/tap).

5.3. Obsługa dotyku i gestów przez Input System

Obsługa dotyku przez nowe API EnhancedTouch

```
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.EnhancedTouch;

public class NewInputTouchDemo : MonoBehaviour
{
    void OnEnable()
    {
        // Włączenie rozszerzonego API dotyku
        EnhancedTouchSupport.Enable();
        TouchSimulation.Enable(); // Symulacja dotyku z myszy (edytor)
```

```

    }

    void OnDisable()
    {
        EnhancedTouchSupport.Disable();
    }

    void Update()
    {
        var touches = UnityEngine.InputSystem.EnhancedTouch.Touch.activeTouches;

        foreach (var touch in touches)
        {
            if (touch.phase == UnityEngine.InputSystem.TouchPhase.Began)
            {
                Debug.Log($"IS Touch #{touch.touchId} @ {touch.screenPosition}");
            }
        }
    }
}

```

5.4. Odczyt akcelerometru przez Input System

Odczyt akcelerometru i żyroskopu przez nowy Input System

```

using UnityEngine;
using UnityEngine.InputSystem;

public class NewInputAccel : MonoBehaviour
{
    void Update()
    {
        // Sprawdzenie dostepnosci akcelerometru
        var accel = Accelerometer.current;
        if (accel == null) return;

        // Odczyt wartosci
        Vector3 acceleration = accel.acceleration.ReadValue();
        Debug.Log($"Akcelerometr: {acceleration}");

        // Żyroskop
        var gyro = AttitudeSensor.current;
        if (gyro != null)
        {
            Quaternion attitude = gyro.attitude.ReadValue();
            transform.rotation = attitude;
        }
    }
}

```

- Dokumentacja Unity: [Input System - dokumentacja](#)
- Dokumentacja Unity: [Enhanced Touch API](#)

7. Bezpieczny obszar ekranu: Safe Area i Notch

Współczesne smartfony często posiadają elementy fizyczne zasłaniające fragment ekranu: wcięcie (notch), otwór na kamerkę (punch-hole) lub wyspę dynamiczną (Dynamic Island). Renderowane treści nie powinny być za nimi ukryte. Unity udostępnia pole `Screen.safeArea` zwracające prostokąt bezpiecznego obszaru w pikselach.

6.1. Co to jest Safe Area

Screen.safeArea to obiekt **Rect** wyznaczający obszar ekranu wolny od przeszkód systemowych. Na typowym telefonie z wcięciem: **safeArea.yMin > 0** (margines od góry)

i **safeArea.yMax < Screen.height** (margines od dołu dla paska nawigacji). Na starszych urządzeniach bez wcięcia safeArea pokrywa cały ekran.

6.2. Skrypt SafeAreaFitter

Standardowym podejściem jest stworzenie skryptu, **który** dopasowuje RectTransform głównego Canvas lub kontenera UI do granic Safe Area. Skrypt dodajemy do obiektu dziecka wewnątrz Canvas (nie do samego Canvas).

Skrypt SafeAreaFitter dopasowujący kontener UI do bezpiecznego obszaru

```
using UnityEngine;

/// <summary>
/// Dopasowuje RectTransform rodzica do bezpiecznego obszaru ekranu.
/// Dodaj ten skrypt do pustego obiektu-kontenera wewnątrz Canvas.
/// </summary>
[RequireComponent(typeof(RectTransform))]
public class SafeAreaFitter : MonoBehaviour
{
    private RectTransform _rectTransform;
    private Rect _lastSafeArea;
    private ScreenOrientation _lastOrientation;

    void Awake()
    {
        _rectTransform = GetComponent<RectTransform>();
        _lastSafeArea = Rect.zero;
        _lastOrientation = Screen.orientation;
        ApplySafeArea();
    }

    void Update()
    {
        // Odświeżaj tylko przy zmianie SafeArea lub orientacji
        if (Screen.safeArea != _lastSafeArea ||
            Screen.orientation != _lastOrientation)
        {
            ApplySafeArea();
        }
    }

    private void ApplySafeArea()
    {
        _lastSafeArea = Screen.safeArea;
        _lastOrientation = Screen.orientation;

        Rect safeArea = Screen.safeArea;
        Vector2 screenSize = new Vector2(Screen.width, Screen.height);

        // Normalizacja do zakresu 0..1 (anchory RectTransform)
        Vector2 anchorMin = safeArea.position / screenSize;
        Vector2 anchorMax = (safeArea.position + safeArea.size) / screenSize;

        _rectTransform.anchorMin = anchorMin;
        _rectTransform.anchorMax = anchorMax;
        _rectTransform.offsetMin = Vector2.zero;
        _rectTransform.offsetMax = Vector2.zero;

        Debug.Log($"SafeArea zastosowana: {safeArea} na {screenSize}");
    }
}
```

WSKAZÓWKA

Hierarchia: Canvas > SafeAreaContainer (skrypt SafeAreaFitter) > reszta **elementów** UI. Tylko kontener powinien mieć dopasowywany RectTransform. Elementy tła (RawImage z tłem aplikacji) pozostają poza kontenerem, aby wypełniać cały ekran łącznie z obszarami systemowymi.

- Dokumentacja Unity: [Screen.safeArea](#)
- Dokumentacja Unity: [Unity Safe Area - poradnik społeczności](#)

8. Orientacja ekranu i responsywność UI

Aplikacje mobilne mogą działać w orientacji pionowej (Portrait), poziomej (Landscape) lub obydwu. Prawidłowe zarządzanie orientacją jest kluczowe dla komfortu użytkownika i poprawnego wyświetlania interfejsu.

7.1. Ustawienia orientacji w Player Settings

Podstawową konfigurację orientacji ustawiamy w **Edit > Project Settings > Player > Resolution and Presentation > Default Orientation**. Dostępne opcje:

Opcja	Znaczenie
Portrait	Wyłącznie pionowa
Portrait Upside Down	Pionowa odwrócona (rzadko używana)
Landscape Left	Pozioma, przycisk Home po lewej
Landscape Right	Pozioma, przycisk Home po prawej
Auto Rotation	Automatycznie dopasowuje się do fizycznego obrotu

7.2. Programowa zmiana orientacji

Programowa zmiana i blokada orientacji ekranu

```
using UnityEngine;

public class OrientationManager : MonoBehaviour
{
    // Zablokowanie orientacji na portrait
    public void LockPortrait()
    {
        Screen.orientation = ScreenOrientation.Portrait;
    }

    // Zablokowanie orientacji na landscape
    public void LockLandscape()
    {
        Screen.orientation = ScreenOrientation.LandscapeLeft;
    }

    // Przywrócenie automatycznej rotacji
    public void EnableAutoRotation()
    {
        Screen.orientation = ScreenOrientation.AutoRotation;
        Screen.autorotateToPortrait = true;
        Screen.autorotateToPortraitUpsideDown = false;
        Screen.autorotateToLandscapeLeft = true;
        Screen.autorotateToLandscapeRight = true;
    }
}
```

```
}  
}
```

7.3. Responsywny Canvas - Anchor i Layout Groups

Kluczem do responsywnego interfejsu jest prawidłowe użycie systemu Anchor w komponentach RectTransform oraz komponentów Layout Group. Poniżej zebrano kluczowe wskazówki:

- Przyciski i panele powinny mieć anchor dopasowany do krawędzi lub rogu ekranu, a nie do stałej pozycji pikselowej.
- Dla **elementów** rozciąganych na cały ekran ustawiamy anchorMin = (0,0) i anchorMax = (1,1), a offsety zerujemy.
- VerticalLayoutGroup i HorizontalLayoutGroup automatycznie rozmieszczają dzieci z kontrolowanym paddingiem i spacingiem.
- ContentSizeFitter pozwala elementom automatycznie dostosowywać rozmiar do zawartości.

WSKAZÓWKA

Zalecany przepływ pracy przy wielu rozdzielczościach: projektuj dla jednej rozdzielczości referencyjnej (np. 1080x2340 px dla Portrait), testuj w Device Simulator dla kilku popularnych urządzeń (iPhone 15 Pro, Samsung Galaxy S24, iPad Pro) i sprawdzaj czy UI nie wychodzi poza SafeArea.

- Dokumentacja Unity: [Screen.orientation](#)
- Dokumentacja Unity: [Canvas Scaler](#)

9. Optymalizacja wydajności na platformach mobilnych

Urządzenia mobilne mają znacznie mniej **zasobów** obliczeniowych i pamięci niż komputery stacjonarne. Aplikacja musi działać płynnie (minimum 30 FPS, optymalnie 60 FPS) na docelowych **urządzeniach**, jednocześnie nie przeciążając baterii i nie powodując przegrzewania się **urządzenia**. Poniżej omówiono najważniejsze techniki optymalizacji.

8.1. Profiler. diagnozowanie problemów

Pierwszym krokiem jest zawsze diagnostyka. Unity Profiler (Window > Analysis > Profiler) umożliwia rejestrowanie wydajności aplikacji w czasie rzeczywistym. Kluczowe zakładki to:

Zakładka Profiler	Co mierzy
CPU Usage	Czas klatki, dominujące skrypty, GC Alloc (alokacje smieci)
GPU Usage	Czas renderowania, overdraw, fill rate
Memory	Zużycie RAM, alokacje zasobów, wycieki pamięci
Rendering	Liczba Draw Calls, SetPass Calls, trojkątów, wierzchołków

Aby profilować na fizycznym urządzeniu, należy połączyć je kablem USB, włączyć tryb debugowania w opcjach dewelopera systemu Android i w edytorze Unity w zakładce Profiler nacisnąć ikonkę samochodu i wybrać podłączone urządzenie.

8.2. CPU - optymalizacja skryptów

Najczęstsze przyczyny problemów z CPU w aplikacjach mobilnych Unity:

- Nadużywanie GetComponent() w Update() - należy cachować referencje w Awake()/Start().
- Nadmierne alokacje w Update() prowadzące do częstych uruchomien Garbage Collector (GC). Należy unikać new, LINQ, konkatenacji stringów w gorących ścieżkach.
- Użycie metod Find() (GameObject.Find, FindObjectOfType) w Update() - są bardzo wolne.
- Nadmierne wywołania fizyki - zmniejsz Fixed Timestep lub ogranicz Collision Layers.

Porównanie złej i dobrej praktyki cachowania komponentów

```
// ZŁE - GetComponent wywoływane co klatkę
void Update()
{
    GetComponent<Rigidbody2D>().velocity = Vector2.zero; // koszt co klatkę!
}

// DOBRZE - referencja cachowana raz w Awake()
private Rigidbody2D _rb;
void Awake() { _rb = GetComponent<Rigidbody2D>(); }
void Update() { _rb.linearVelocity = Vector2.zero; }
```

8.3. GPU - optymalizacja renderowania

Draw Call Batching

Każdy Draw Call to polecenie dla procesora GPU: „narysuj ten mesh z tym materiałem”. Ich nadmierna liczba spowalnia renderowanie. Unity oferuje dwie techniki redukcji:

- Static Batching - łączy statyczne obiekty (Static flag) w jeden mesh. Włączone domyślnie.
- Dynamic Batching - łączy małe dynamiczne meshe o wspólnym materiale. Włączamy w Player Settings.
- GPU Instancing - rysuje wiele instancji tego samego mesha jednym wywołaniem. Włączamy w ustawieniach materiału (Enable GPU Instancing).

Tekstury i kompresja

Tekstury są głównym zużyciem pamięci GPU. Na platformach mobilnych stosujemy:

Format kompresji	Platforma / zastosowanie
ASTC	iOS (A8+) i Android (ARM Mali, Adreno 4xx+) - zalecany
ETC2	Android (OpenGL ES 3.0) - dobry współczynnik jakości/rozmiar
DXT (BC)	Desktop Windows/Linux - nie na mobile!

Mipmaps: zawsze włączaj dla tekstur używanych w przestrzeni 3D. Dla sprite'ów UI można wyłączyć (Generate Mip Maps = false) jeżeli rozmiar na ekranie jest stały.

8.4. Pamięć i zasoby

- Asynchroniczne ładowanie zasobów przez Addressables lub Resources.LoadAsync() zapobiega skokowym zatrzymaniom klatki podczas wczytywania.

- Object Pooling - zamiast Instantiate/Destroy tworzenie puli **obiektów** do wielokrotnego użytku (szczególnie dla efektów cząsteczkowych i pocisków).
- Kompresja dźwięku: muzyka tła jako Streaming (Vorbis/MP3), **krótkie** SFX jako Decompress On Load (ADPCM).

8.5. Limity klatek i energooszczędność

Ustawienia limitu klatek i usypiania ekranu

```
void Start()
{
    // Ograniczenie do 60 FPS (oszczędność baterii)
    Application.targetFrameRate = 60;

    // Wylaczenie synchronizacji pionowej (vSync) na mobile - nie działa tak jak na PC
    QualitySettings.vSyncCount = 0;

    // Wylaczenie usypiania ekranu podczas gry
    Screen.sleepTimeout = SleepTimeout.NeverSleep;
}
```

- Dokumentacja Unity: [Unity Profiler](#)
- Dokumentacja Unity: [Optymalizacja dla Android](#)
- Dokumentacja Unity: [Object Pooling](#)

10. Budowanie i wdrożenie aplikacji (Build & Deploy)

Koncowym etapem tworzenia aplikacji mobilnej jest wygenerowanie pliku instalacyjnego i jego wdrożenie na urządzenie lub opublikowanie w sklepie. W tej sekcji skupiamy się na platformie Android jako bardziej dostępnej w środowisku laboratoryjnym, z **krótkim** opisem specyfiki platformy iOS.

9.1. Przygotowanie środowiska Android

Aby można było budować aplikacje na Androida, w Unity Hub musi być zainstalowany moduł Android Build Support zawierający Android SDK, NDK i OpenJDK. Sprawdzamy to w Unity Hub > Installs > klikamy ikonkę koła zębatego przy wersji edytora > Add Modules.

Wymagany moduł	Opis
Android Build Support	Bazowy moduł wymagany do budowania APK/AAB
Android SDK & NDK Tools	Narzędzia deweloperskie Google (automatycznie z modelem)
OpenJDK	Środowisko Java wymagane przez Gradle

9.2. Player Settings dla Androida

Otwieramy **File > Build Settings > Android > Player Settings**. Najważniejsze pola do skonfigurowania:

Ustawienie	Opis i zalecana wartość
Company Name / Product Name	Identyfikuje aplikację w sklepie i systemie
Package Name	Unikalny identyfikator odwrotnej domeny: com.firma.aplikacja
Version	Numer wersji widoczny dla użytkownika (np. 1.0.0)

Bundle Version Code	Liczba całkowita zwiększana przy każdej publikacji
Minimum API Level	Minimalna wersja Androida: Android 8.0 (API 26) zalecane
Target API Level	Aktualna wymagana przez Google Play: Android 14 (API 34)
Scripting Backend	IL2CPP (wymagany dla Google Play + lepsze czasy uruchomienia)
Target Architectures	ARM64 (obowiązkowe dla Google Play od 2019), opcjonalnie ARMv7
Default Icon	Ikona aplikacji (zalecane 1024x1024 px PNG)
Splash Screen	Ekran startowy aplikacji

9.3. Podpisywanie aplikacji (Keystore)

Każda aplikacja na Androida musi być podpisana kluczem kryptograficznym (keystore). Bez podpisu nie można opublikować aplikacji w Google Play. W Player Settings > Publishing Settings możemy:

- Stworzyć nowy keystore (Create a new keystore) - przy pierwszym projekcie.
- Użyć istniejącego keystore - obowiązkowe przy aktualizacji aplikacji w sklepie. Zmiana keystore powoduje traktowanie aplikacji jako nowej!
- Podać alias i hasło - potrzebne przy **Każdym** procesie budowania.

UWAGA

Keystore to plik o krytycznym znaczeniu. Jego utrata uniemożliwia aktualizację aplikacji w Google Play. Należy go bezwzględnie zachować w bezpiecznym miejscu i wykonać kopie zapasową. Nie przechowywać keystore w repozytorium Git!

9.4. Budowanie pliku APK / AAB

Unity umożliwia generowanie **dwóch** formatów plików Android:

- APK (Android Package) - tradycyjny format, może być bezpośrednio zainstalowany na urządzeniu (adb install) lub przesłany użytkownikowi. Zalecany do testów.
- AAB (Android App Bundle) - format wymagany przez Google Play od 2021 roku. Sklep sam generuje z niego optymalne APK dla każdego **urządzenia**.

Instalacja APK przez Android Debug Bridge (adb)

```
# Urządzenie musi być podłączone USB z włączonym USB Debugging

# Sprawdzenie podłączonych urządzeń
adb devices

# Instalacja pliku APK
adb install -r ścieżka/do/aplikacja.apk

# Uruchomienie aplikacji (opcjonalne)
adb shell am start -n com.firma.aplikacja/.UnityPlayerActivity
```

Rys. 17. Instalacja APK przez narzędzie adb w terminalu

9.5. Debugowanie na urządzeniu fizycznym

Po włączeniu USB Debugging na urządzeniu możemy łączyć się z nim przez Unity Profiler i okno Console. Dzienniki Debug.Log() są widoczne przez:

- Unity Console - po podłączeniu **urządzenia** i wyborze go jako źródła logów w dolnym lewym rogu okna Console.
- Android Studio Logcat - pełny dostęp do logów systemowych i Unity.
- adb logcat -s Unity - filtrowanie logów Unity w terminalu.

9.6. Specyfika platformy iOS

Budowanie na iOS wymaga komputera Mac z zainstalowanym Xcode (wersja aktualna). Unity generuje projekt Xcode, **który** następnie należy skompilować i podpisać certyfikatem deweloperskim Apple. Wymagane jest konto Apple Developer (99 USD/rok dla dystrybucji w App Store).

- W Player Settings > iOS ustawiamy Bundle Identifier (analogicznie do Package Name).
- Kamera, mikrofon i inne zasoby systemowe wymagają wpisania opisu użycia w polach Camera Usage Description, Microphone Usage Description itd.
- Testowanie na urządzeniu iOS bez płatnego konta możliwe jest lokalnie przez Xcode (Free Provisioning) z limitami (np. brak push notifications).
- Dokumentacja Unity: [Build Settings - Android](#)
- Dokumentacja Unity: [Google Play - wymagania AAB](#)
- Dokumentacja Unity: [Publishing for iOS](#)

11. Zadania dla studenta

Poniższe zadania należy wykonać samodzielnie na bazie projektu z poprzednich laboratoriów (Unity 1 i Unity 2). Efektem każdego zadania powinno być działające rozwiązanie przetestowane w edytorze (Device Simulator) i opcjonalnie na fizycznym urządzeniu Android.

1. Implementacja obsługi dotyku: po dotknięciu ekranu obiekt 2D (np. prostokąt) ma pojawiać się w miejscu dotknięcia. Przy przesunięciu palcem obiekt ma podążać za palcem (TouchPhase.Moved). Przy uniesieniu palca obiekt powinien zaniknąć.
2. Implementacja detektora gestu Swipe: przełączanie widocznych paneli za pomocą swipe w lewo i prawo (podobnie do karuzeli na stronie internetowej). Minimum dwa panele z różną zawartością.
3. Sterowanie głośnością muzyki tła za pomocą gestu Pinch: rozsuniecie palców zwiększać głośność, zbliżenie - zmniejszać. Wartość głośności wyświetlać na bieżąco w polu Text-MeshPro.
4. Integracja akcelerometru: stworzenie mini-gry labiryntowej, w której kulka 2D (Rigidbody2D) jest sterowana przez pochylenie **urządzenia**. Kulka nie powinna wychodzić poza granice labiryntu (Collider2D na ścianach).
5. Implementacja skryptu SafeAreaFitter i przetestowanie aplikacji w Device Simulator dla co najmniej 3 różnych urządzeń z wcięciem (iPhone 15 Pro, Samsung Galaxy S24, Pixel 8 Pro). Udokumentować różnice w wyglądzie interfejsu przed i po zastosowaniu skryptu.

6. Budowanie i instalacja aplikacji na fizycznym urządzeniu Android (APK). Przetestowanie działania dotyku, gestów i akcelerometru na urządzeniu. Profilowanie aplikacji przez Unity Profiler podłączony do **urządzenia** USB i zanotowanie liczby Draw Calls oraz średniego czasu klatki (ms).

12. Zadania dodatkowe

Zadania dodatkowe przeznaczone są dla studentów chcących rozszerzyć wiedzę poza program obowiązkowy. Ocena aktywności laboratoryjnej może uwzględnić realizację poniższych zadań.

- Implementacja globalnego systemu haptic feedback: wibracja krótka (przycisk UI) i długa (błąd, koniec gry) z użyciem `Handheld.Vibrate()` i natywnego API Android (`AndroidJavaClass`). Na iOS należy skorzystać z Taptic Engine przez natywny plugin.
- Rozszerzenie detektora gestów o obsługę gestu obrotu dwoma palcami (Rotation Gesture): mierzenie zmiany kąta między wektorem łączącym dwa palce w kolejnych klatkach i stosowanie jej do obrotu wybranego obiektu 2D.
- Implementacja systemu powiadomień lokalnych (Local Notifications) przez pakiet Unity Mobile Notifications (dostępny w Package Manager). Powiadomienie powinno pojawiać się 10 sekund po zamknięciu aplikacji z treścią personalizowaną imieniem gracza zapisanym w `PlayerPrefs`.
- Stworzenie ekranu podsumowania wyposażonym w animowany wykres słupkowy renderowany przez skrypt C# z użyciem API GL lub `LineRenderer`, prezentujący 5 najlepszych wyników zapisanych w `PlayerPrefs`.
- Konfiguracja Addressables (Package Manager > Addressables) dla grafik tła i muzyki: zasoby ładowane asynchronicznie tylko gdy są potrzebne i zwalniane po wyjściu ze sceny. Porównanie zużycia pamięci RAM przed i po wdrożeniu Addressables.

Studenci realizujący zadania dodatkowe mogą korzystać z poniższych zasobów:

- Dokumentacja Unity: [Unity Mobile Notifications](#)
- Dokumentacja Unity: [Addressables - Getting Started](#)
- Dokumentacja Unity: [Android Haptics](#)
- Dokumentacja Unity: [Unity GL - immediate mode graphics](#)

ĆWICZENIE LABORATORYJNE - UNITY 4

Architektura danych · dynamiczne UI · kamera powiadomienia · lokalizacja · komunikacja sieciowa

Programowanie Aplikacji Mobilnych - moduł Unity

1. Spis treści

1. Wprowadzenie i przegląd zagadnień	49
2. ScriptableObjects, czyli architektura oparta na danych.....	50
3. Komunikacja sieciowa - UnityWebRequest i REST API	52
4. Serializacja JSON: JsonUtility i Newtonsoft.Json.....	54
5. Dynamiczne listy UI: ScrollView i prefab spawning	56
6. Dostęp do kamery przez WebCamTexture	57
7. Lokalizacja aplikacji, czyli Unity Localization	59
8. Unity Mobile Notifications, czyli powiadomienia lokalne.....	61
9. Wzorce architektoniczne MVC i MVP w Unity	64

2. Wprowadzenie i przegląd zagadnień

Pierwsze trzy laboratoria pozwoliły zapoznać się ze środowiskiem Unity, podstawami interfejsu użytkownika, nawigacją między scenami, animacjami, korutynami, obsługą dotyku i czujnikami urządzenia. Niniejsza instrukcja przenosi projekt na poziom aplikacji produktowej - zdolnej pobierać dane z Internetu, wyświetlać je w dynamicznie generowanym interfejsie oraz korzystać z **zasobów** systemowych telefonu takich jak kamera. Omówione zostaną również kluczowe kwestie związane z lokalizacją na różne języki i powiadomieniami lokalnymi.

Po wykonaniu ćwiczeń opisanych w tej instrukcji student będzie potrafił:

- projektować architekturę danych z użyciem ScriptableObjects,
- wysyłać zadania HTTP GET i POST do zewnętrznych API,
- deserializować odpowiedzi JSON do obiektów C#,
- dynamicznie generować elementy listy UI na podstawie pobranych danych,
- odczytywać obraz z kamery urządzenia i wyświetlać go w interfejsie,
- dodać obsługę wielu języków przez pakiet Unity Localization,
- planować i wywoływać powiadomienia lokalne na Androidzie i iOS,

- stosować wzorce MVC i MVP w architekturze aplikacji Unity.

3. ScriptableObjects, czyli architektura oparta na danych

ScriptableObject to klasa bazowa Unity pozwalająca tworzyć zasoby danych niezależnie od scen i obiektów gry (GameObjects). Instancje ScriptableObject są przechowywane jako pliki assetów w projekcie (.asset) i mogą być współdzielone przez wiele komponentów bez tworzenia kopii danych w pamięci.

2.1. Dlaczego ScriptableObjects zamiast MonoBehaviour

Tradycyjne podejście do przechowywania danych konfiguracyjnych (np. statystyki postaci, ustawienia poziomów, definicje elementów sklepu) polega na tworzeniu publicznych pól w MonoBehaviour lub używaniu statycznych klas. Obydwa podejścia mają wady:

- MonoBehaviour z danymi musi być przypięte do obiektu na scenie - dane są powiązane ze sceną i nie są łatwo współdzielone.
- Klasy statyczne nie są serializowane przez edytor Unity i utrudniają testowanie.
- PlayerPrefs sprawdza się tylko dla prostych wartości prymitywnych.

ScriptableObjects rozwiązują te problemy: są widoczne w Inspektorze, łatwo dzielone pomiędzy scenami, modyfikowalne bez zmiany kodu i idealnie nadają się jako kontenery konfiguracji. Odpowiadają wzorcowi projektowemu **Data Object**.

2.2. Tworzenie ScriptableObject

Klasa ScriptableObject dziedziczy po UnityEngine.ScriptableObject i jest dekorowana atrybutem [CreateAssetMenu] ułatwiającym tworzenie instancji z menu edytora.

ProduktSO.cs - Definicja klasy ScriptableObject dla produktu w aplikacji sklepowej	C#
<pre>using UnityEngine; // Atrybut tworzy wpis w menu: Assets > Create > Dane > Definicja Produktu [CreateAssetMenu(fileName = "NowyProdukt", menuName = "Dane/Definicja Produktu", order = 0)] public class ProduktSO : ScriptableObject { [Header("Podstawowe informacje")] public string nazwaRynkowa; public string opis; public Sprite ikona; public float cena; [Header("Parametry techniczne")] public int poziomTrudnosci; // 1-5 public bool dostepnyOffline; [Header("Statystyki (tylko runtime)")] [System.NonSerialized] // Nie zapisywane do pliku public int liczbaPobran; }</pre>	

Po zapisaniu skryptu w oknie Project klikamy prawym przyciskiem myszy i wybieramy **Create > Dane > Definicja Produktu**. Powstaje plik .asset z wszystkimi polami widocznymi i edytowalnymi w Inspektorze, bez konieczności uruchamiania aplikacji.

Rys. 2. Instancja ScriptableObject widoczna i edytowalna w oknie Inspektora

2.3. Użycie ScriptableObject w MonoBehaviour

ProduktWidok.cs	C#
<pre> using UnityEngine; using TMPro; using UnityEngine.UI; /// <summary> /// Wyświetla dane produktu z przypisanego ScriptableObject. /// Zmiana danych w assetach SO natychmiast odzwierciedla się w UI. /// </summary> public class ProduktWidok : MonoBehaviour { [SerializeField] private ProduktSO dane; [SerializeField] private TextMeshProUGUI txtNazwa; [SerializeField] private TextMeshProUGUI txtOpis; [SerializeField] private TextMeshProUGUI txtCena; [SerializeField] private Image imgIkona; void Start() => OdswiezWidok(); public void OdswiezWidok() { if (dane == null) return; txtNazwa.text = dane.nazwaRynkowa; txtOpis.text = dane.opis; txtCena.text = \$"{dane.cena:F2} PLN"; imgIkona.sprite = dane.ikona; } } </pre>	

2.4. ScriptableObject jako kanał zdarzeń (event channel)

Zaawansowanym zastosowaniem ScriptableObjects jest budowanie systemu zdarzeń niezależnych od scen. Zamiast statycznych zdarzeń C# tworzymy assetowe kanały zdarzeń, które mogą być współdzielone między dowolnymi scenami i komponentami.

ZdarzenieProsteSO.cs - ScriptableObject jako assetowy kanał zdarzeń luźno łączący sceny	C#
<pre> using UnityEngine; using UnityEngine.Events; // Asset reprezentujący kanał zdarzenia bez parametrow [CreateAssetMenu(menuName = "Zdarzenia/Zdarzenie Proste")] public class ZdarzenieProsteSO : ScriptableObject { private UnityAction _listeners; public void Wywolaj() => _listeners?.Invoke(); public void Subskrybuj(UnityAction cb) => _listeners += cb; public void Odsubskrybuj(UnityAction cb) => _listeners -= cb; } // — Nadawca zdarzenia (np. przycisk w menu) ————— public class NadawcaZdarzenia : MonoBehaviour { [SerializeField] private ZdarzenieProsteSO zdarzenie; public void Wywolaj() => zdarzenie.Wywolaj(); } // — Odbiornik zdarzenia (np. panel ekwipunku w innej scenie) ————— public class OdbiornikZdarzenia : MonoBehaviour { [SerializeField] private ZdarzenieProsteSO zdarzenie; void OnEnable() => zdarzenie.Subskrybuj(NaZdarzenie); void OnDisable() => zdarzenie.Odsubskrybuj(NaZdarzenie); void NaZdarzenie() => Debug.Log("Zdarzenie odebrane!"); } </pre>	

WSKAZOWKA

Wzorzec SO Event Channel jest promowany przez 'Overthrowing the MonoBehaviour Tyranny in a Glorious Scriptable Object Revolution' (Ryan Hipple, Unite Austin 2017). Eliminuje konieczność wyszukiwania referencji między scenami przez Find().

- Dokumentacja: [ScriptableObject - dokumentacja](#)
- Dokumentacja: [SO Architecture - Ryan Hipple \(GH\)](#)

4. Komunikacja sieciowa: UnityWebRequest i REST API

Większość współczesnych aplikacji mobilnych komunikuje się z serwerami przez interfejsy REST API oparte na protokole HTTP. Odpowiedzi serwera są najczęściej sformatowane w JSON lub XML. Unity udostępnia klasę UnityWebRequest do asynchronicznego wysyłania zapytań HTTP bez blokowania głównego wątku aplikacji.

3.1. Podstawy http: metody i kody odpowiedzi

Metoda HTTP	Zastosowanie
GET	Pobieranie danych (lista produktów , profil użytkownika)
POST	Tworzenie zasobu lub wysyłanie formularza (rejestracja, komentarz)
PUT / PATCH	Aktualizacja istniejącego zasobu
DELETE	Usunięcie zasobu

Kod odpowiedzi	Znaczenie
200 OK	Sukces - zawartosc w body odpowiedzi
201 Created	Zasob został utworzony (odpowiedz na POST)
400 Bad Request	Błędne zapytanie (np. brakujące pole)
401 Unauthorized	Brak lub niepoprawny token uwierzytelniający
404 Not Found	Zasob nie istnieje
500 Internal Server Error	Błąd po stronie serwera

3.2. Pobieranie danych zapytaniem HTTP GET

UnityWebRequest jest używany wewnątrz korutyny. Instrukcja yield return wstrzymuje wykonanie do momentu otrzymania odpowiedzi, nie blokując głównego wątku i nie zawieszając UI aplikacji.

```

ApiKlient.cs C#
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

public class ApiKlient : MonoBehaviour {
    private const string URL_BAZA = "https://jsonplaceholder.typicode.com";

    void Start() {
        StartCoroutine(PobierzPosty());
    }

    IEnumerator PobierzPosty() {
        string url = $"{URL_BAZA}/posts";

        using (var request = UnityWebRequest.Get(url)) {
            // Nagłówek identyfikujący klienta
        }
    }
}

```

```

        request.SetRequestHeader("Accept", "application/json");

        // Oczekiwanie na odpowiedz
        yield return request.SendWebRequest();

        // Obsługa błędu sieciowego
        if (request.result != UnityWebRequest.Result.Success) {
            Debug.LogError($"Błąd HTTP: {request.responseCode} | {request.error}");
            yield break;
        }

        // Sukces - treść odpowiedzi jako string
        string json = request.downloadHandler.text;
        Debug.Log($"Odpowiedz ({request.responseCode}): {json.Length} znaków");

        // Dalsze przetwarzanie JSON - patrz sekcja 4
        PrzetworzOdpowiedz(json);
    }

    void PrzetworzOdpowiedz(string json) { /* sekcja 4 */ }
}

```

Rys. 5. Zapytanie HTTP GET z pełną obsługą błędów w korutynie

3.3. Wysyłanie danych zapytaniem HTTP POST

WyslijPost.cs - Zapytanie HTTP POST z body JSON i nagłówkiem Content-Type	C#
<pre> using UnityEngine; using UnityEngine.Networking; using System.Collections; IEnumerator WyslijPost(string tytuł, string tresc) { string url = \$"{URL_BAZA}/posts"; // Tworzenie obiektu do serializacji var nowyPost = new { title = tytuł, body = tresc, userId = 1 }; string bodyJson = JsonUtility.ToJson(nowyPost); // Dane jako bajty UTF-8 byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(bodyJson); using (var request = new UnityWebRequest(url, "POST")) { request.uploadHandler = new UploadHandlerRaw(bodyRaw); request.downloadHandler = new DownloadHandlerBuffer(); request.SetRequestHeader("Content-Type", "application/json"); yield return request.SendWebRequest(); if (request.result == UnityWebRequest.Result.Success) { Debug.Log(\$"Utworzono: {request.responseCode}\n{request.downloadHandler.text}"); } else { Debug.LogError(\$"POST nie powiodł się: {request.error}"); } } } </pre>	

3.4. Uwierzytelnianie Bearer Token

Większość publicznych API wymaga uwierzytelniania. Standardowym mechanizmem jest nagłówek Authorization z tokenem Bearer (np. JWT):

Dodawanie nagłówków uwierzytelniających do zapytania HTTP

C#

```
// Nagłówek uwierzytelniający Bearer Token
using UnityEditor.PackageManager.Requests;

request.SetRequestHeader("Authorization", $"Bearer {tokenJWT}");

// Przykład: uwierzytelnianie kluczem API
request.SetRequestHeader("X-API-Key", kluczApi);
```

UWAGA

Nigdy nie przechowuj kluczy API, haseł ani tokenów bezpośrednio w kodzie źródłowym lub plikach projektu w repozytorium Git. Użyj zmiennych środowiskowych, zewnętrznego vault (np. Unity Cloud, AWS Secrets Manager) lub obfuskacji przez narzędzia takie jak dotenv. Wyciek klucza w publicznym repozytorium może prowadzić do nieuprawnionego użycia i kosztów rozliczeniowych.

- Dokumentacja: [UnityWebRequest - dokumentacja](#)
- Dokumentacja: [JSONPlaceholder - testowe REST API](#)

5. Serializacja JSON: JsonUtility i Newtonsoft.Json

JSON (JavaScript Object Notation) to lekki format wymiany danych powszechnie stosowany w REST API. Unity oferuje wbudowane narzędzie JsonUtility, a przez Package Manager dostępna jest znacznie potężniejsza biblioteka Newtonsoft.Json (Json.NET), będąca standardem w ekosystemie .NET.

4.1. Wbudowane narzędzie JsonUtility

JsonUtility działa z klasami oznaczonymi atrybutem [System.Serializable]. Jest szybkie i nie wymaga dodatkowych pakietów, jednak ma istotne ograniczenia: nie obsługuje słowników (Dictionary), kolekcji interfejsów, typów nullable ani tablic najwyższego poziomu (JSON zaczynający się od []).

Serializacja i deserializacja JSON przez JsonUtility z obejściem tablicy

C#

```
using System;
using UnityEngine;

[Serializable]
public class PostDto
{
    public int userId;
    public int id;
    public string title;
    public string body;
}

// Deserializacja pojedynczego obiektu
PostDto post = JsonUtility.FromJson<PostDto>(jsonString);
Debug.Log($"Post #{post.id}: {post.title}");

// Serializacja do JSON
string json = JsonUtility.ToJson(post, prettyPrint: true);

// — Obejście dla tablicy JSON („[...]”) - wrapper —————
[Serializable]
public class PostListWrapper
{
    public PostDto[] posts;
}
```

```
// Przed uzyciem zawijamy json: „{\"posts\": \" + json + \"}”
string opakowany = "{\"posts\": \" + jsonTablica + \"}”;
var lista = JsonUtility.FromJson<PostListaWrapper>(opakowany);
```

4.2. Newtonsoft.Json: pełne wsparcie JSON

Instalacja: Window > Package Manager > wyszukaj **com.unity.nuget.newtonsoft-json** lub dodaj do pliku Packages/manifest.json: **"com.unity.nuget.newtonsoft-json": "3.2.1"**.

ProduktWidok.cs	C#
<pre>using Newtonsoft.Json; using System.Collections.Generic; using System.Xml; // Deserializacja tablicy JSON bezpośrednio List<PostDto> posty = JsonConvert.DeserializeObject<List<PostDto>>(jsonString); Debug.Log(\$"Pobrano {posty.Count} postow"); // Deserializacja z opcjami (tolerancja na null, błędy) var ustawienia = new JsonSerializerSettings { NullValueHandling = NullValueHandling.Ignore, MissingMemberHandling = MissingMemberHandling.Ignore, }; List<PostDto> postyBezp = JsonConvert.DeserializeObject<List<PostDto>>(jsonString, ustawienia); // Obsługa słownika (nie możliwa w JsonUtility) Dictionary<string, int> statystyki = JsonConvert.DeserializeObject<Dictionary<string, int>>(jsonSłownik); // Serializacja z wcięcia (prettified) string ladnyJson = JsonConvert.SerializeObject(posty, Formatting.Indented);</pre>	

Rys. 9. Newtonsoft.Json, deserializacja list, słowników i tolerancja błędów

Cecha	JsonUtility	Newtonsoft.Json
Wymagania	Brak (wbudowany)	Pakiet NuGet
Tablice na szczycie JSON	Nie (wrapper)	Tak
Dictionary<K,V>	Nie	Tak
Nullable types (int?)	Nie	Tak
Wydajność	Bardzo wysoka	Wysoka
Zalecany do	Prostych modeli SO	API produkcyjnych

WSKAZOWKA

Dla projektów produkcyjnych zalecane jest Newtonsoft.Json. JsonUtility wystarczy dla prostych modeli danych, gdzie struktura JSON jest własna i kontrolowana.

- Dokumentacja: [JsonUtility](#)
- Dokumentacja: [Newtonsoft.Json dla Unity](#)

6. Dynamiczne listy UI: ScrollView i prefab spawning

Aplikacje mobilne często wyświetlają listy elementów o nieznanej z góry długości: wyniki wyszukiwania, produkty sklepowe, wiadomości, komentarze. Unity oferuje komponent ScrollRect (ScrollView) z dynamicznie generowanymi elementami-prefabami jako standardowe rozwiązanie dla tego wzorca.

5.1. Budowa hierarchii ScrollView

Standardowa hierarchia obiektu listy w Canvas:

Hierarchia komponentów ScrollView dla listy pionowej	C#
<pre>using UnityEngine; Canvas ScrollView ← GameObject > UI > Scroll View Viewport ← Mask + Image (obcina zawartosc) Content ← Vertical Layout Group + Content Size Fitter Scrollbar Vertical</pre>	

Content musi posiadać dwa kluczowe komponenty:

- VerticalLayoutGroup (lub HorizontalLayoutGroup) - automatycznie rozmieszcza dzieci w kolumnie. Włączamy Control Child Size: Width i Height oraz Child Force Expand.
- ContentSizeFitter - automatycznie dopasowuje rozmiar Content do sumy rozmiarów dzieci. Ustawiamy Vertical Fit = Preferred Size.

5.2. Prefab elementu listy

Tworzenie prefabu elementu listy:

- Projektujemy jeden element UI w scenie (np. panel z obrazkiem, tytułem i opisem).
- Przeciągamy go do folderu Prefabs w oknie Project - powstaje plik .prefab.
- Usuwamy oryginalny obiekt ze sceny - prefab będzie instancjonowany programowo.

Skrypt prefabu elementu listy z metoda inicjalizacji danymi	C#
<pre>using UnityEngine; using UnityEngine.UI; using TMPro; /// <summary> /// Skrypt przypisany do prefabu pojedynczego elementu listy produktow. /// Odpowiada za wyswietlanie danych jednego produktu. /// </summary> public class ElementListy : MonoBehaviour { [SerializeField] private TextMeshProUGUI txtNazwa; [SerializeField] private TextMeshProUGUI txtCena; [SerializeField] private Image imgIkona; [SerializeField] private Button btnSzczegoly; private ProduktSO _dane; public void Inicjalizuj(ProduktSO dane) { _dane = dane; txtNazwa.text = dane.nazwaRynkowa; txtCena.text = \$"{dane.cena:F2} PLN"; imgIkona.sprite = dane.ikona; btnSzczegoly.onClick.RemoveAllListeners(); btnSzczegoly.onClick.AddListener(NaSzczegoly); } }</pre>	

```
private void NaSzczegoly() {
    Debug.Log($"Otwieranie szczegółów dla: {_dane.nazwaRynkowa}");
}
}
```

5.3. Dynamiczne generowanie listy

KontrolerListy.cs - Dynamiczne tworzenie i czyszczenie elementów listy ScrollView	C#
<pre>using System.Collections.Generic; using Unity.Android.Gradle.Manifest; using UnityEngine; public class KontrolerListy : MonoBehaviour { [SerializeField] private GameObject prefabElementu; [SerializeField] private Transform kontenerContent; [SerializeField] private List<ProduktSO> produkty; // Przywoływane po pobraniu danych z API lub przy starcie public void OdswiezListe(List<ProduktSO> noweDane) { // 1. Usuniecie starych elementów for (int i = kontenerContent.childCount - 1; i >= 0; i--) { Destroy(kontenerContent.GetChild(i).gameObject); } // 2. Instancjonowanie nowych elementów foreach (var produkt in noweDane) { var instancja = Instantiate(prefabElementu, kontenerContent); instancja.GetComponent<ElementListy>().Inicjalizuj(produkt); } } void Start() => OdswiezListe(produkty); }</pre>	

5.4. Optymalizacja; wirtualizacja listy

Przy długich listach (100+ elementów) Instantiate/Destroy każdego elementu jest kosztowne. Stosujemy wtedy techniki wirtualizacji:

- Object Pooling - recykling instancji prefabów (opisany w Lab 3).
- Unity UI Toolkit (UI Elements) - nowy system UI z wbudowaną wirtualizacją ListView (zalecany dla projektów na Unity 2022+).
- External assets: Recyclable Scroll Rect, Smooth Scroll Rect z Asset Store.

WSKAZOWKA

Dla list do około 50-100 elementów standardowy ScrollView z VerticalLayoutGroup jest w pełni wystarczający na urządzeniach mobilnych. Optymalizacja przez Object Pool staje się konieczna dopiero przy setkach widocznych elementów.

- Dokumentacja: [ScrollRect](#)
- Dokumentacja: [VerticalLayoutGroup](#)
- Dokumentacja: [UI Toolkit ListView](#)

7. Dostęp do kamery przez WebCamTexture

Smartfony posiadają kamerę przednią i tylną, które mogą być używane jako źródło obrazu w aplikacji Unity. Typowe zastosowania to: skanery QR/kodów kreskowych, filtry AR, zdjęcia profilowe, aplikacje rzeczywistości rozszerzonej. Unity udostępniła klasę WebCamTexture

umożliwiająca strumieniowanie obrazu z kamery do tekstury użytej na materiałach lub elementach UI.

6.1. Uprawnienia systemowe

Przed użyciem kamery aplikacja musi posiadać odpowiednie uprawnienia systemowe. Na Androidzie deklaruje się je automatycznie przez Unity, lecz wymagana jest zgoda użytkownika w czasie działania aplikacji (runtime permission):

Platforma	Wymagane uprawnienie	Gdzie konfiguruje się
Android	android.permission.CAMERA	Automatycznie - Unity dodaje do AndroidManifest.xml
iOS	NSCameraUsageDescription (Info.plist)	Player Settings > iOS > Camera Usage Description

UWAGA

Aplikacja na iOS bez wypełnionego pola NSCameraUsageDescription zostanie odrzucona przez App Store Review. Opis musi jasno wyjaśnić użytkownikowi po co aplikacja potrzebuje dostępu do kamery, np. 'Aplikacja używa kamery do skanowania kodów QR.'

6.2. Strumieniowanie obrazu z kamery

PodgladKamery.cs - Strumieniowanie obrazu z tylnej kamery do elementu RawImage

C#

```
using UnityEngine;
using UnityEngine.UI;

public class PodgladKamery : MonoBehaviour {
    [SerializeField] private RawImage podgladObraz;
    [SerializeField] private AspectRatioFitter dopasowanie;

    private WebCamTexture _kameraTextura;

    void Start() {
        // Wyświetlenie dostępnych kamer
        foreach (var urzadzenie in WebCamTexture.devices)
            Debug.Log($"Kamera: {urzadzenie.name} | Przednia: {urzadzenie.isFrontFacing}");

        // Wybór kamery tylnej (isFrontFacing == false)
        WebCamDevice kameraTylna = System.Array.Find(
            WebCamTexture.devices,
            d => !d.isFrontFacing
        );

        // Inicjalizacja strumienia: szerokosc x wysokosc x fps
        _kameraTextura = new WebCamTexture(
            kameraTylna.name, 1280, 720, 30);

        podgladObraz.texture = _kameraTextura;
        _kameraTextura.Play();
    }

    void Update() {
        // Korekcja proporcji obrazu po jego inicjalizacji
        if (_kameraTextura.width > 16 && dopasowanie != null) {
            dopasowanie.aspectRatio =
                (float)_kameraTextura.width / _kameraTextura.height;
        }
    }
}
```

```

void OnDestroy() {
    // ZAWSZE zatrzymaj kamere i zwolnij zasob
    if (_kameraTextura != null && _kameraTextura.isPlaying) {
        _kameraTextura.Stop();
        Destroy(_kameraTextura);
    }
}
}

```

6.3. Przechwytywanie klatki (zdjecie)

Przechwytywanie klatki z WebCamTexture i zapis do pliku PNG	C#
<pre> using UnityEngine; /// <summary> /// Przechwytyuje biezaca klatke obrazu kamery jako Texture2D. /// Moze byc uzyta do zapisu zdjecia lub dalszego przetwarzania. /// </summary> public Texture2D ZrobZdjecie() { if (_kameraTextura == null !_kameraTextura.isPlaying) { Debug.LogWarning("Kamera nie jest aktywna!"); return null; } // Skopiowanie aktualnej klatki do nowej tekstury Texture2D zdjecie = new Texture2D(_kameraTextura.width, _kameraTextura.height, TextureFormat.RGB24, mipChain: false); zdjecie.SetPixels(_kameraTextura.GetPixels()); zdjecie.Apply(); // Zapis do galerii (wymaga Android Permission WRITE_EXTERNAL_STORAGE // lub MediaStore API na Android 10+) byte[] pngBytes = zdjecie.EncodeToPNG(); string sciezka = System.IO.Path.Combine(Application.persistentDataPath, "zdjecie.png"); System.IO.File.WriteAllBytes(sciezka, pngBytes); Debug.Log(\$"Zdjecie zapisane: {sciezka}"); return zdjecie; } </pre>	

WSKAZOWKA

WebCamTexture.GetPixels() jest operacja CPU-bound. Dla aplikacji wymagających przetwarzania obrazu w czasie rzeczywistym (np. detekcja twarzy) zalecane jest użycie Unity Barracuda (inferencja ML) lub natywnych bibliotek przez Android/iOS plugin (np. OpenCV for Unity z Asset Store).

- Dokumentacja: [WebCamTexture](#)
- Dokumentacja: [Permissions Android](#)
- Dokumentacja: [NativeCamera \(Asset Store\)](#)

8. Lokalizacja aplikacji, czyli Unity Localization

Lokalizacja (L10n) to proces dostosowania aplikacji do różnych języków i kultur. Obejmuje nie tylko tłumaczenie tekstów, ale również formatowanie liczb, dat, walut, kierunek czytania tekstu (LTR/RTL) oraz zmianę **zasobow** (grafik, dźwięków) dla konkretnych regionów. Unity oferuje dedykowany pakiet Localization.

7.1. Instalacja i konfiguracja

Pakiet instalujemy przez **Window > Package Manager > Unity Registry > Localization**. Po instalacji pojawia się menu **Edit > Project Settings > Localization**, gdzie definiujemy wszystkie obsługiwane języki (Locales).

Podstawowa konfiguracja krok po kroku:

- Edit > Project Settings > Localization > Create > wybieramy języki (np. Polish, English, German).
- W oknie Localization Tables (Window > Asset Management > Localization Tables) tworzymy tablice napisów: Create > String Table Collection, np. o nazwie UI_Texty.
- W tablicy dodajemy wpisy (klucze) i ich tłumaczenia dla każdego języka.
- W ustawieniach projektu wybieramy język domyślny (Default Locale).

Rys. 15. Okno Localization Tables z wpisami dla języka polskiego i angielskiego

7.2. Użycie lokalizowanego tekstu w UI

Komponent **Localize String Event** (dostępny po instalacji pakietu) doczepiamy do obiektu TextMeshPro i konfigurujemy w Inspektorze: wybieramy tablice i klucz napisu oraz opcjonalnie łączy je ze zdarzeniem aktualizującym pole tekstowe. Alternatywnie można korzystać ze skryptu:

MenedzerLokalizacji.cs - Programowe pobieranie lokalizowanych tekstów i zmiana języka	C#
<pre> using UnityEngine; using UnityEngine.Localization; using UnityEngine.Localization.Settings; using UnityEngine.Localization.Tables; using TMPro; using System.Collections; public class MenedzerLokalizacji : MonoBehaviour { [SerializeField] private TextMeshProUGUI txtTytul; [SerializeField] private TextMeshProUGUI txtWitaj; void Start() { StartCoroutine(WczytajTeksty()); } IEnumerator WczytajTeksty() { // Oczekiwanie na inicjalizację systemu lokalizacji yield return LocalizationSettings.InitializationOperation; // Pobranie napisu z tablicy var tytul = LocalizationSettings.StringDatabase .GetLocalizedString("UI_Texty", "TYTUL_APLIKACJI"); var witaj = LocalizationSettings.StringDatabase .GetLocalizedString("UI_Texty", "POWITANIE"); txtTytul.text = tytul; txtWitaj.text = witaj; } // Zmiana języka w czasie działania aplikacji public IEnumerator ZmienJęzyk(string kodJęzyka) { var locale = LocalizationSettings.AvailableLocales .GetLocale(new LocaleIdentifier(kodJęzyka)); if (locale != null) { LocalizationSettings.SelectedLocale = locale; yield return LocalizationSettings.InitializationOperation; Debug.Log(\$"Język zmieniony na: {locale.LocaleName}"); } } } </pre>	

```

    }
}
}

```

7.3. Lokalizacja zasobów (Sprite, AudioClip)

Poza tekstami można lokalizować dowolne zasoby Unity: obrazy, dźwięki, czcionki. W oknie Localization Tables wybieramy **Asset Table Collection** i przypisujemy różne wartości dla każdego języka. Komponenty **Localize Sprite Event** i **Localize Audio Clip Event** działają analogicznie jak Localize String Event.

7.4. Smart Strings, czyli napisy z parametrami

Pakiet Localization obsługuje Smart Strings, czyli napisy z dynamicznymi wartościami wstawianymi w czasie działania aplikacji. Przykład: 'Witaj, {imie}! Masz {liczba} wiadomości.'

Smart Strings, napisy z dynamicznie podstawianymi parametrami	C#
<pre> // Klucz w tablicy: POWITANIE_DYNAMICZNE // Wartosc PL: "Witaj, {imie}! Masz {liczba} wiadomosci." // Wartosc EN: "Hello, {imie}! You have {liczba} messages." using System.Collections.Generic; using UnityEngine; var parametry = new Dictionary<string, object> { { "imie", PlayerPrefs.GetString("PlayerName", "Graczu") }, { "liczba", 5 } }; string napis = LocalizationSettings.StringDatabase .GetLocalizedString("UI_Texty", "POWITANIE_DYNAMICZNE", parametry); // Wynik PL: "Witaj, Jan! Masz 5 wiadomości." </pre>	

- Dokumentacja: [Unity Localization - dokumentacja](#)
- Dokumentacja: [Smart Strings](#)

9. Unity Mobile Notifications, czyli powiadomienia lokalne

Powiadomienia lokalne (local notifications) to komunikaty generowane przez aplikację i wyświetlane przez system operacyjny, nawet gdy aplikacja działa w tle lub jest zamknięta. Typowe zastosowania: przypomnienia o codziennym logowaniu, powiadomienie o zakończeniu zadania, alerty o zdarzeniu po upływie czasu.

8.1. Instalacja pakietu

Instalujemy pakiet przez **Window > Package Manager > Unity Registry > Mobile Notifications**. Pakiet instaluje wersje dla Androida (NotificationCenter) i iOS (UserNotifications). Po instalacji w Player Settings pojawia się sekcja **Mobile Notifications Settings** gdzie konfigurujemy kanały powiadomień.

8.2. Konfiguracja kanałów Androida

Android 8.0+ wymaga zdefiniowania kanałów powiadomień przed ich wysyłaniem. Kanał grupuje powiadomienia o podobnym celu i pozwala użytkownikowi zarządzać nimi w ustawieniach systemu.

AndroidKanalPowiadomien.cs - Rejestracja kanałów powiadomień Android z różnym priorytetem	C#
---	----

```

using Unity.Notifications.Android;
using UnityEngine;

public static class AndroidKanalPowiadomien {
    public const string ID_GLOWNY = "kanal_glowny";
    public const string ID_PROMOCJE = "kanal_promocje";

    public static void Inicjalizuj() {
        // Kanal główny - wysoki priorytet
        var kanalGlowny = new AndroidNotificationChannel {
            Id = ID_GLOWNY,
            Name = "Powiadomienia główne",
            Description = "Ważne powiadomienia z aplikacji",
            Importance = Importance.High,
        };
        AndroidNotificationCenter.RegisterNotificationChannel(kanalGlowny);

        // Kanal promocji - niski priorytet
        var kanalPromocje = new AndroidNotificationChannel {
            Id = ID_PROMOCJE,
            Name = "Promocje i oferty",
            Description = "Informacje o okazjach i rabatach",
            Importance = Importance.Low,
        };
        AndroidNotificationCenter.RegisterNotificationChannel(kanalPromocje);
    }
}

```

8.3. Wysyłanie powiadomienia na Androidzie

MenedzerPowiadomien.cs - Planowanie, wysyłanie i anulowanie powiadomień lokalnych Android	C#
<pre> using Unity.Notifications.Android; using System; using UnityEngine; public class MenedzerPowiadomien : MonoBehaviour { void Start() { // Inicjalizacja kanałów przy starcie AndroidKanalPowiadomien.Inicjalizuj(); } /// <summary> /// Planuje powiadomienie z opóźnieniem podanym w minutach. /// </summary> public void ZaplanujPowiadomienie(string tytuł, string tresc, int opoznienieMin) { var powiadomienie = new AndroidNotification { Title = tytuł, Text = tresc, SmallIcon = "default", LargeIcon = "icon_aplikacji", FireTime = DateTime.Now.AddMinutes(opoznienieMin), RepeatInterval = null, // null = jednorazowe }; int idPowiadomienia = AndroidNotificationCenter.SendNotification(powiadomienie, AndroidKanalPowiadomien.ID_GLOWNY); Debug.Log(\$"Zaplanowano powiadomienie #{idPowiadomienia} za {opoznienieMin} min"); } /// <summary> /// Anuluje konkretne zaplanowane powiadomienie. /// </summary> public void AnulujPowiadomienie(int id) { AndroidNotificationCenter.CancelNotification(id); } } </pre>	

```

        Debug.Log($"Anulowano powiadomienie #{id}");
    }

    /// <summary>
    /// Anuluje wszystkie zaplanowane powiadomienia aplikacji.
    /// </summary>
    public void AnulujWszystkie() {
        AndroidNotificationCenter.CancelAllNotifications();
    }
}

```

8.4. Prośba o uprawnienia do powiadomienia na iOS

Na iOS przed wysłaniem jakiegokolwiek powiadomienia konieczne jest uzyskanie zgody użytkownika. Brak zgody powoduje ciche pomijanie powiadomień. Prośba wyświetla się jednorazowo:

ProduktWidok.cs - Prośba o uprawnienia i zaplanowanie powiadomienia na platformie iOS	C#
<pre> #if UNITY_IOS using Unity.Notifications.iOS; IEnumerator PoprosPOUprawnieniaiOS() { var prosba = new AuthorizationRequest(AuthorizationOption.Alert AuthorizationOption.Badge Authorization- Option.Sound, registerForRemoteNotifications: false); yield return prosba; if (prosba.IsError) { Debug.LogError(\$"Bład uprawnień: {prosba.Error}"); } else if (prosba.Granted) { Debug.Log("Użytkownik zgodził się na powiadomienia"); ZaplanujPowiadomienieiOS("Hej!", "Nie zapomnij o logowaniu dziś!", 60); } else { Debug.Log("Użytkownik odrzucił powiadomienia"); } } void ZaplanujPowiadomienieiOS(string tytuł, string tresc, int opoznienieSek) { var trigger = new iOSNotificationTimeIntervalTrigger { TimeInterval = TimeSpan.FromSeconds(opoznienieSek), Repeats = false }; var powiadomienie = new iOSNotification { Title = tytuł, Body = tresc, Trigger = trigger, ShowInForeground = false, }; iOSNotificationCenter.ScheduleNotification(powiadomienie); } #endif </pre>	

UWAGA

Nigdy nie wysyłaj powiadomień bez zgody użytkownika. Natrętne powiadomienia są głównym powodem odinstalowania aplikacji. Poproszenie o zgodę powinno nastąpić po zaprezentowaniu użytkownikowi wartości powiadomień w sposób kontekstowy, a nie przy pierwszym uruchomieniu aplikacji.

- Dokumentacja: [Unity Mobile Notifications](#)
- Dokumentacja: [Android Notification Channels](#)

10. Wzorce architektoniczne MVC i MVP w Unity

Wzorce architektoniczne pomagają organizować kod w sposób czytelny, testowalny i łatwy w rozbudowie. Bez świadomej architektury projekty Unity szybko przekształcają się w zestaw skryptów MonoBehaviour o silnie splątanej logice (tzw. 'Big Ball of Mud'). W tej sekcji omówimy wzorec MVC (Model-View-Controller) i jego wariant MVP (Model-View-Presenter) dostosowany do środowiska Unity.

9.1. Model-View-Controller (MVC)

MVC dzieli aplikację na trzy warstwy:

- **Model:** dane i logika biznesowa (ScriptableObject, klasy POCO, serwisy sieciowe). Nie zna Widoku ani Kontrolera.
- **View (widok):** elementy UI (TextMeshPro, Image, Button). Odpowiada za prezentację. Nie zawiera logiki biznesowej.
- **Controller (kontroler):** pośrednik łączący Model z Widokiem. Reaguje na zdarzenia użytkownika, aktualizuje Model i odświeża Widok.

Rys. 21. Schemat zależności w wzorcu MVC zastosowanym w Unity

UzytkownikWidok.cs - Przykład implementacji wzorca MVC dla profilu użytkownika	C#
<pre>// — MODEL — klasa POCO, brak zależności od Unity UI — using TMPro; using UnityEngine; [System.Serializable] public class UzytkownikModel { public string Imie; public int Punkty; public int Poziom => Punkty / 100; // Obliczana właściwość } // — VIEW — czysty MonoBehaviour, tylko wyświetlanie — public class UzytkownikWidok : MonoBehaviour { [SerializeField] public TextMeshProUGUI txtImie; [SerializeField] public TextMeshProUGUI txtPunkty; [SerializeField] public TextMeshProUGUI txtPoziom; [SerializeField] public Button btnDodajPunkty; public void Aktualizuj(UzytkownikModel model) { txtImie.text = model.Imie; txtPunkty.text = \$"Punkty: {model.Punkty}"; txtPoziom.text = \$"Poziom: {model.Poziom}"; } } // — CONTROLLER — łączy Model z Widokiem, obsługuje zdarzenia — public class UzytkownikKontroler : MonoBehaviour { [SerializeField] private UzytkownikWidok widok; private UzytkownikModel _model; void Start() { _model = new UzytkownikModel { Imie = "Jan", Punkty = 0 }; widok.Aktualizuj(_model); widok.btnDodajPunkty.onClick.AddListener(DodajPunkty); } private void DodajPunkty() { _model.Punkty += 10; // Modyfikacja Modelu widok.Aktualizuj(_model); // Aktualizacja Widoku // Zapis do PlayerPrefs lub wysłanie do API PlayerPrefs.SetInt("Punkty", _model.Punkty); } }</pre>	

9.2. Model-View-Presenter (MVP)

MVP to ewolucja MVC, w której Presenter (zamiast Kontrolera) posiada bezpośrednie referencje do interfejsów Widoku i Modelu, a Widok jest możliwie prosty i pasywny. W Unity MVP jest szczególnie użyteczny przy testowaniu jednostkowym, gdyż Presenter nie dziedziczy po `MonoBehaviour` i może być testowany bez sceny Unity.

UzytkownikPresenter.cs - Implementacja wzorca MVP z interfejsem `IUzytkownikWidok` dla testowalnego Presentera C#

```
// Interfejs Widoku - Presenter nie zna konkretnej klasy UI
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public interface IUzytkownikWidok {
    void UstawImie(string imie);
    void UstawPunkty(int punkty);
    void UstawPoziom(int poziom);
    event System.Action NaDodajPunkty;
}

// Presenter - NIE dziedziczy po MonoBehaviour, łatwo testowalny
public class UzytkownikPresenter {
    private readonly IUzytkownikWidok _widok;
    private readonly UzytkownikModel _model;

    public UzytkownikPresenter(IUzytkownikWidok widok, UzytkownikModel model) {
        _widok = widok;
        _model = model;

        _widok.NaDodajPunkty += ObsluzDodajPunkty;
        OdswiezWidok();
    }

    private void ObsluzDodajPunkty() {
        _model.Punkty += 10;
        OdswiezWidok();
    }

    private void OdswiezWidok() {
        _widok.UstawImie(_model.Imie);
        _widok.UstawPunkty(_model.Punkty);
        _widok.UstawPoziom(_model.Poziom);
    }
}

// MonoBehaviour implementujace interfejs - most miedzy Unity a Presenterem
public class UzytkownikWidokMB : MonoBehaviour, IUzytkownikWidok {
    [SerializeField] private TextMeshProUGUI txtImie;
    [SerializeField] private TextMeshProUGUI txtPunkty;
    [SerializeField] private TextMeshProUGUI txtPoziom;
    [SerializeField] private Button btnDodajPunkty;

    public event System.Action NaDodajPunkty;
    private UzytkownikPresenter _presenter;

    void Start() {
        var model = new UzytkownikModel { Imie = "Jan", Punkty = 0 };
        _presenter = new UzytkownikPresenter(this, model);
        btnDodajPunkty.onClick.AddListener(() => NaDodajPunkty?.Invoke());
    }

    public void UstawImie(string imie) => txtImie.text = imie;
    public void UstawPunkty(int p) => txtPunkty.text = $"Punkty: {p}";
    public void UstawPoziom(int l) => txtPoziom.text = $"Poziom: {l}";
}
```

Cecha	MVC w Unity	MVP w Unity
Testowanie Kontrolera	Trudne (<code>MonoBehaviour</code>)	Łatwe (zwykła klasa C#)
Widok	Aktywny (reaguje na zdarzenia)	Pasywny (tylko interfejs)

Sprzężenie Widok-logika	Pośrednie (Kontroler)	Ściśle przez interfejs
Złożoność kodu	Niska-średnia	Średnia-wysoka
Zalecany do	Prostych ekranów i paneli	Złożonych ekranów z testami

- Dokumentacja: [Game Programming Patterns - R. Nystrom \(darmowa online\)](#)
- Dokumentacja: [Wprowadzenie do MVC w Unity](#)

11. Zadania dla studenta

Poniższe zadania należy wykonać samodzielnie, tworząc nowy projekt lub rozszerzając projekt z poprzednich laboratoriów. Każde zadanie powinno być przetestowane w edytorze i opcjonalnie na fizycznym urządzeniu Android.

1. Projektowanie architektury danych: stworzenie co najmniej dwóch różnych klas Scriptable-Object (np. ProduktSO i KategoriaSO), wypełnienie ich danymi przez Inspektor i wyświetlenie w panelu UI. Zmiana danych w assetach powinna być natychmiast widoczna w uruchomionej aplikacji.
2. Integracja z REST API: pobranie listy wpisów z publicznego API (<https://jsonplaceholder.typicode.com/posts>), deserializacja JSON i wyświetlenie 10 pierwszych wyników w dynamicznie generowanej liście ScrollView. Każdy element listy powinien zawierać tytuł i fragment treści. Dodać wskaźnik ładowania (Loading Spinner lub pasek postępu) widoczny w trakcie oczekiwania na odpowiedź serwera.
3. Podgląd kamery i zdjęcie: stworzenie ekranu aparatu wyświetlającego obraz z tylnej kamery **urządzenia** z przyciskami Zrob Zdjecie i Przełącz Kamere (przednia/tylna). Wykonane zdjęcie powinno być wyświetlone w osobnym panelu obok podglądu na żywo.
4. Lokalizacja UI: dodanie obsługi co najmniej **dwóch** języków (np. polskiego i angielskiego) do głównego ekranu aplikacji. Wszystkie napisy przycisków, tytułów i komunikatów powinny być pobierane z tablicy lokalizacji. Dodać selektor języka (Dropdown lub dwa przyciski flagi) zmieniający język w czasie działania aplikacji.
5. Powiadomienie lokalne: po naciśnięciu przycisku Zaplanuj Przypomnienie aplikacja powinna zaplanować powiadomienie z treścią pobraną z pola tekstowego InputField, wyświetlane po 1 minucie od zamknięcia aplikacji. Dodać przycisk Anuluj usuwający zaplanowane powiadomienie.
6. Refaktoryzacja kodu: wybranie jednego ekranu z projektu i przepisanie go z użyciem wzorca MVC lub MVP. Kontroler lub Presenter nie może zawierać bezpośrednich referencji do komponentów UI (TextMeshPro, Image, Button). Udokumentowanie wprowadzonych zmian w komentarzach XML.

12. Zadania dodatkowe

Zadania dodatkowe wymagają samodzielnego przeszukania dokumentacji i **zasobów** zewnętrznych. Stanowią rozszerzenie programu dla ambitnych studentów i mogą być uwzględnione przy ocenie aktywności laboratoryjnej.

- Skanowanie kodów QR: integracja biblioteki ZXing.Net (dostępnej przez NuGet / Asset Store) z podglądem WebCamTexture. Wykryty kod QR powinien zostać zdekodowany

i jego zawartość wyświetlona w polu tekstowym. Jeśli kod zawiera poprawny adres URL - otworzyć go w przeglądarce przez `Application.OpenURL()`.

- Autoryzacja OAuth2: implementacja prostego przepływu logowania przez Google Sign-In (Unity plugin) lub Firebase Authentication. Po zalogowaniu wyświetlić imię i awatar użytkownika pobrane z profilu Google. Token sesji przechowywać bezpiecznie (szyfrowanie przez `SecurePlayerPrefs` lub zewnętrzny plugin).
- Testowanie jednostkowe Presentera: stworzenie projektu testowego Unity Test Framework (Window > General > Test Runner) z co najmniej 5 testami weryfikującymi logikę Presentera z zadania 6. Testy powinny używać mocków interfejsu `IWidok` zamiast rzeczywistych komponentów UI.
- Obsługa błędu sieci z kolejkowaniem: rozbudowanie klienta REST API o mechanizm kolejki zadań. Jeżeli urządzenie jest offline (`Application.internetReachability == NetworkReachability.NotReachable`), zadania HTTP powinny być zapisywane do kolejki w `PlayerPrefs` i automatycznie ponawiane przy powrocie połączenia.
- Własny `ScriptableObject` Event Bus: implementacja generycznego kanału zdarzeń `SO` obsługującego dowolny typ parametru przez generyki C# (`ZdarzenieSOGeneric<T>`). Demonstracja na przykładzie zdarzenia zmiany wyniku (int) i zdarzenia zmiany profilu użytkownika (`UzytkownikModel`). Napisanie odpowiadających testów jednostkowych weryfikujących kolejność i liczbę wywołań.

Przydatne zasoby do zadań dodatkowych:

- Dokumentacja: [ZXing.Net QR decoder](#)
- Dokumentacja: [Unity Test Framework](#)
- Dokumentacja: [Firebase Unity SDK](#)
- Dokumentacja: [Application.internetReachability](#)

ĆWICZENIE LABORATORYJNE - UNITY 5

AR Foundation · wykrywanie płaszczyzn · śledzenie obrazów, twarzy · zakotwiczenie przestrzenne

Programowanie Aplikacji Mobilnych - moduł Unity

1. Spis treści

1. Wprowadzenie do rzeczywistości rozszerzonej	68
2. AR Foundation - instalacja i konfiguracja projektu	69
3. Struktura sceny AR: XR Origin i AR Session	71
4. Wykrywanie płaszczyzn (plane detection)	72
5. Umieszczanie obiektów przez Raycast AR	74
6. Śledzenie obrazów (image tracking)	76
7. Śledzenie twarzy (face tracking)	78
8. Zakotwiczenie przestrzenne (AR Anchors)	79
9. Optymalizacja i testowanie aplikacji AR	81

2. Wprowadzenie do rozszerzonej rzeczywistości

Rzeczywistość rozszerzona (ang. Augmented Reality, AR) to technologia nakładania wirtualnych elementów graficznych na obraz ze świata rzeczywistego rejestrowany przez kamerę urządzenia. W odróżnieniu od rzeczywistości wirtualnej (VR), w AR użytkownik pozostaje w kontakcie z otoczeniem fizycznym, a elementy cyfrowe wzbogacają jego percepcję. Przykładami zastosowań AR w życiu codziennym są: nawigacja z naniesieniem tras na obraz kamery, przymiarki mebli w pomieszczeniu, filtry twarzy w mediach społecznościowych, instrukcje serwisowe w przemyśle oraz gry mobilne.

1.1. Podstawowe pojęcia

Pojęcie	Definicja
Znacznik (marker)	Obraz lub obiekt rozpoznawany przez system AR jako punkt odniesienia
Markerless AR	AR bez znaczników - oparty na analizie środowiska (płaszczyzny, głębia)
Płaszczyzna AR	Wykryta powierzchnia pozioma lub pionowa, na której można umieszczać obiekty

Zakotwica (anchor)	Punkt w przestrzeni śledzony przez system AR, do którego przypinamy obiekt
Estymacja oświetlenia	Szacowanie warunków świetlnych sceny na potrzeby renderowania cieni i odbłasków
6DoF (Degrees of Freedom)	Swoboda ruchu urządzenia w trzech osiach położenia i trzech osiach obrotu

1.2. Platformy i frameworki AR

Wiodące platformy AR na urządzeniach mobilnych to ARCore (Android) i ARKit (iOS). Oba systemy dostarczają zbliżony zestaw funkcji, lecz korzystają z oddzielnych API. Unity AR Foundation stanowi warstwę abstrakcji umożliwiającą pisanie jednej bazy kodu działającej na obu platformach.

Platforma	System	Wymagania sprzętowe
ARCore	Android	Android 9.0+ z ARCore-compatible SoC (większość flagowców od 2018 r.)
ARKit	iOS	iOS 13+ na iPhone 6s lub nowszym, pełne 6DoF od iPhone X
AR Foundation	Oba	Wersja 5.x dla Unity 6 (6000.x); obsługuje ARCore i ARKit przez jedno API

1.3. Lista obsługiwanych funkcji AR Foundation 5.x

Poszczególne funkcje AR Foundation mogą być niedostępne na starszych urządzeniach lub systemach. Przed użyciem każdej funkcji należy programowo sprawdzić, czy dana podsystemu jest obsługiwana przez aktywną platformę.

Funkcja AR Foundation	ARCore	ARKit
Wykrywanie płaszczyzn poziomych	Tak	Tak
Wykrywanie płaszczyzn pionowych	Tak	Tak
Śledzenie obrazów (marker AR)	Tak	Tak
Śledzenie twarzy	Ograniczone	Tak (TrueDepth)
Zakotwiczenie przestrzenne	Tak	Tak
Siatki świata (World Meshes)	Nie	Tak (LiDAR)
Okkluzja przez głębokość	Wybrane urz.	Tak (LiDAR)
Współdzielone sesje AR	Tak	Tak

- Dokumentacja: [ARCore - lista urządzeń obsługiwanych](#)
- Dokumentacja: [ARKit - wymagania sprzętowe](#)

3. AR Foundation - instalacja i konfiguracja projektu

AR Foundation nie jest domyślnie zainstalowany w Unity. Wymaga również osobnych pakietów dostawców: ARCore XR Plugin dla Androida oraz ARKit XR Plugin dla iOS. Całość instalujemy przez Unity Package Manager.

2.1. Instalacja pakietów

Otwieramy okno Package Manager (Window > Package Manager), przechylamy filtr na Unity Registry i instalujemy kolejno następujące pakiety.

Pakiet	Wersja (Unity 6)	Przeznaczenie
AR Foundation	6.0.x	Wspólne API AR dla wszystkich platform
Google ARCore XR Plugin	6.0.x	Implementacja AR dla systemu Android
Apple ARKit XR Plugin	6.0.x	Implementacja AR dla systemu iOS
XR Plugin Management	4.x	Zarządzanie podsystemami XR (wymagany)

UWAGA

Numer wersji pakietów muszą być zgodne. Mieszanie różnych wersji AR Foundation i pluginów platformowych (np. AR Foundation 6.0 z ARCore 5.0) powoduje błędy kompilacji. Należy zawsze instalować zestawy o identycznych numerach wersji głównych.

2.2. Konfiguracja XR Plugin Management

Po zainstalowaniu pakietów przechodzimy do Edit > Project Settings > XR Plug-in Management. W zakładce Android włączamy opcję ARCore, a w zakładce iOS opcję ARKit. System automatycznie aktywuje odpowiedni podsystem przy uruchomieniu aplikacji na danej platformie.

Rys. 1. Okno XR Plug-in Management z włączonym ARCore (Android) i ARKit (iOS)

2.3. Ustawienia Player Settings dla AR

Aplikacje AR wymagają dodatkowej konfiguracji w Player Settings (Edit > Project Settings > Player).

Platforma	Ustawienie	Zalecana wartość
Android	Minimum API Level	Android 9.0 (API 28)
Android	Scripting Backend	IL2CPP
Android	Target Architectures	ARM64 (obowiązkowe)
Android	Graphics API	OpenGL ES3 lub Vulkan
iOS	Camera Usage Description	Opis po polsku lub angielsku (wymagany)
iOS	Minimum iOS Version	13.0
iOS	Architecture	ARM64

2.4. Konfiguracja jakości renderowania dla AR

Aplikacje AR renderują obraz kamery jako tło, co wiąże się z dodatkowym kosztem GPU. Zaleca się użycie profilu jakości dostosowanego do platform mobilnych. W Edit > Project Settings > Quality tworzymy nowy profil AR Mobile z następującymi ustawieniami:

- Pixel Light Count: 1 (ograniczenie światła w czasie rzeczywistym)

- Anisotropic Textures: Per Texture (nie Forced On)
- Anti Aliasing: 2x Multi Sampling (kompromis jakości i wydajności)
- Shadows: Hard Shadows Only lub Disable (najsilniejszy wpływ na wydajność)
- Realtime Reflection Probes: wyłączone

WSKAZOWKA

Aplikacja AR zawsze wymaga dostępu do kamery. Należy wypełnić pole Camera Usage Description w Player Settings dla iOS, opisując w języku zrozumiałym dla użytkownika, dlaczego kamera jest potrzebna. Brak opisu skutkuje odrzuceniem aplikacji przez App Store Review.

- Dokumentacja: [AR Foundation - Getting Started](#)
- Dokumentacja: [XR Plugin Management](#)

4. Struktura sceny AR: XR Origin i AR Session

Każda scena AR wymaga dwóch kluczowych obiektów: AR Session zarządzającego sesją śledzenia oraz XR Origin definiującego początek układu współrzędnych sceny w odniesieniu do rzeczywistego świata. Unity dostarcza gotowe prefabrykaty tych obiektów.

3.1. Szybkie tworzenie sceny AR

Nową scenę AR tworzymy przez menu: GameObject > XR > AR Session oraz GameObject > XR > XR Origin (Mobile AR). Unity automatycznie tworzy hierarchię obiektów:

Domyślna hierarchia obiektów w scenie AR Foundation	
AR Session	<-- Zarządza życiem sesji śledzenia
AR Input Manager	<-- Odczytuje dane z czujników urządzenia
XR Origin	<-- Główny punkt odniesienia świata AR
Camera Offset	<-- Przesunięcie kamery względem początku
Main Camera	<-- Kamera renderująca z komponentem AR Camera Manager
TrackablesParent	<-- Rodzic dla wszystkich wykrytych obiektów śledzonych

3.2. Komponent AR Session

AR Session kontroluje cykl życia sesji AR: uruchomienie śledzenia, wstrzymanie i zakończenie. Kluczowe właściwości komponentu to.

Właściwość AR Session	Opis
Attempt Update	Włącza próbę aktualizacji ARCore/ARKit do nowszej wersji przy starcie
Match Frame Rate	Synchronizuje częstotliwość klatek Unity z częstotliwością kamery AR

KontrolerSesjiAR.cs - Sprawdzenie obsługi AR na urządzeniu przed uruchomieniem sesji C#

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;

/// <summary>
/// Kontroler sesji AR sprawdzający dostępność AR na urządzeniu.
/// </summary>
public class KontrolerSesjiAR : MonoBehaviour
{
    [SerializeField] private ARSession arSession;
    [SerializeField] private GameObject panelBrakuObslugi;
```

```

async void Start() {
    // Sprawdzenie dostępności AR na urządzeniu
    if (ARSession.state == ARSessionState.None ||
        ARSession.state == ARSessionState.CheckingAvailability) {
        await ARSession.CheckAvailability();
    }

    if (ARSession.state == ARSessionState.Unsupported) {
        Debug.LogWarning("AR nie jest obsługiwane na tym urządzeniu.");
        panelBrakuObslugi.SetActive(true);
        arSession.enabled = false;
    }
    else {
        Debug.Log($"AR dostępne. Stan: {ARSession.state}");
    }
}
}

```

3.3. Komponent XR Origin i AR Camera Manager

XR Origin definiuje początek układu współrzędnych sceny w świecie rzeczywistym. Wszystkie wykryte obiekty AR (płaszczyzny, kotwice, śledzone obrazy) są wyrażane względem tego punktu. Komponent **AR Camera Manager** przyłączony do głównej kamery steruje strumieniem obrazu kamery i udostępnia klatki do analizy przez podsystem platformowy.

OdbiornikKameryAR.cs - Odczyt estymacji oświetlenia z ramki kamery AR dla realistycznego renderowania

C#

```

using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

// Odbiera zdarzenia ramek kamery AR i odczytuje estymację oświetlenia.
public class OdbiornikKameryAR : MonoBehaviour
{
    [SerializeField] private ARCameraManager cameraManager;
    void OnEnable() => cameraManager.frameReceived += NaRamceKamery;
    void OnDisable() => cameraManager.frameReceived -= NaRamceKamery;
    void NaRamceKamery(ARCameraFrameEventArgs args) {
        // Estymacja oświetlenia (jeśli platforma obsługuje)
        if (args.lightEstimation.averageBrightness.HasValue) {
            float jasnoc = args.lightEstimation.averageBrightness.Value;
            RenderSettings.ambientIntensity = jasnoc;
        }

        if (args.lightEstimation.colorCorrection.HasValue) {
            var korekcja = args.lightEstimation.colorCorrection.Value;
            RenderSettings.ambientLight = new Color(
                korekcja.r, korekcja.g, korekcja.b);
        }
    }
}

```

- Dokumentacja: [AR Camera Manager](#)
- Dokumentacja: [XR Origin](#)

5. Wykrywanie płaszczyzn (plane detection)

Wykrywanie płaszczyzn to jedna z podstawowych funkcji AR: system analizuje obraz kamery oraz dane z akcelerometru i żyroskopu, aby rozpoznać płaskie powierzchnie w otoczeniu - podłogi, stoły, ściany. Wykryte płaszczyzny stanowią podstawę do umieszczania obiektów wirtualnych w przestrzeni rzeczywistej.

4.1. Komponent AR Plane Manager

AR Plane Manager dodajemy do obiektu XR Origin. Odpowiada za:

- wykrywanie i aktualizację płaszczyzn w czasie rzeczywistym,
- instancjonowanie wizualizacji płaszczyzn (Plane Prefab),
- wywoływanie zdarzenia planesChanged przy wykryciu, aktualizacji lub usunięciu płaszczyzny.

Tryb wykrywania	Opis
Horizontal Upward	Powierzchnie poziome skierowane ku górze (podłoga, stół)
Horizontal Downward	Powierzchnie poziome skierowane ku dołowi (sufit)
Vertical	Powierzchnie pionowe (ściany, drzwi)
Everything	Wszystkie typy jednocześnie (większy koszt obliczeniowy)
NotTracking	Wykrywanie tymczasowo wyłączone (np. po umieszczeniu obiektu)

4.2. Skrypt obsługi AR Plane Manager

ZaradcaPłaszczyzn.cs - z obsługą dodawania, aktualizacji i usuwania	C#
<pre> using System.Collections.Generic; using UnityEngine; using UnityEngine.XR.ARFoundation; using UnityEngine.XR.ARSubsystems; // Zarządza wykrywaniem płaszczyzn AR i reaguje na ich zmiany. [RequireComponent(typeof(ARPlaneManager))] public class ZaradcaPłaszczyzn : MonoBehaviour { private ARPlaneManager _planeManager; void Awake() => _planeManager = GetComponent<ARPlaneManager>(); void OnEnable() => _planeManager.planesChanged += NaZmieniePłaszczyzn; void OnDisable() => _planeManager.planesChanged -= NaZmieniePłaszczyzn; void NaZmieniePłaszczyzn(ARPlanesChangedEventArgs args) { foreach (var płaszczyzna in args.added) { Debug.Log(\$"Wykryto nową płaszczyznę: {płaszczyzna.trackableId}" + \$" Typ: {płaszczyzna.alignment}"); } foreach (var płaszczyzna in args.updated) { // Płaszczyzna została zaktualizowana (rozszerzone granice) Debug.Log(\$"Zaktualizowano płaszczyznę: {płaszczyzna.trackableId}" + \$" Rozmiar: {płaszczyzna.size}"); } foreach (var płaszczyzna in args.removed) { Debug.Log(\$"Usunięto płaszczyznę: {płaszczyzna.trackableId}"); } } // Ukrywa wizualizacje płaszczyzn po umieszczeniu obiektu. // Wyłącza również dalsze wykrywanie, aby oszczędzić zasoby. public void UkryjPłaszczyzny() { _planeManager.requestedDetectionMode = PlaneDetectionMode.None; foreach (var płaszczyzna in _planeManager.trackables) { płaszczyzna.gameObject.SetActive(false); } } } </pre>	

4.3. Prefabrykant wizualizacji płaszczyzny

Wygląd wykrytej płaszczyzny definiuje prefabrykant przypisany do pola Plane Prefab komponentu AR Plane Manager. Prefabrykant powinien zawierać komponent AR Plane oraz obiekt wizualizacji (np. półprzezroczysty siatka). Unity dostarcza przykładowe prefabrykаты w pakiecie AR Foundation Samples dostępnym na GitHub.

WSKAZOWKA

Po wykryciu wystarczającej liczby płaszczyzn warto wyłączyć dalsze wykrywanie (PlaneDetectionMode.None) oraz ukryć siatki płaszczyzn. Zmniejsza to obciążenie CPU o ok. 15-25% w typowej scenie AR.

- Dokumentacja: [ARPlaneManager](#)
- Dokumentacja: [AR Foundation Samples \(GitHub\)](#)

6. Umieszczanie obiektów przez Raycast AR

Najczęstszym wzorcem interakcji w aplikacjach AR jest tap-to-place: użytkownik dotyka ekranu i aplikacja umieszcza wirtualny obiekt w punkcie płaszczyzny odpowiadającym temu dotknięciu. Realizuje się to przez promień (raycast) wystrzelony ze środka dotknięcia na ekranie w przestrzeń świata AR.

5.1. AR Raycast Manager

AR Raycast Manager dodajemy do obiektu XR Origin obok AR Plane Manager. Udostępnia metodę Raycast() dostosowaną do świata AR, która zwraca punkty trafienia na wykrytych płaszczyznach i zakotwiczeniach.

OdbiornikKameryAR.cs - Umieszczanie i przesuwanie obiektu przez promień AR z obsługą tap-to-place C#

```
using System.Collections.Generic;
using Unity.Physics;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

/// Umieszcza wirtualny obiekt w miejscu dotknięcia ekranu na wykrytej płaszczyźnie.
[RequireComponent(typeof(ARRaycastManager))]
[RequireComponent(typeof(ARPlaneManager))]
public class UmieszczanieObiektów : MonoBehaviour {
    [SerializeField] private GameObject prefabObiektu;
    [SerializeField] private ARPlaneManager planeManager;

    private ARRaycastManager _raycastManager;
    private GameObject _instancjaObiektu;
    private List<ARRaycastHit> _trafienia = new();

    void Awake() {
        _raycastManager = GetComponent<ARRaycastManager>();
    }

    void Update() {
        // Obsługa dotyku przez nowy Input System
        if (Touchscreen.current == null) return;
        var dotyk = Touchscreen.current.primaryTouch;
        if (!dotyk.press.wasPressedThisFrame) return;

        Vector2 pozycjaDotyku = dotyk.position.ReadValue();

        // Wystrzelenie promienia w przestrzeń AR
```

```

bool trafiono = _raycastManager.Raycast(
    pozycjaDotyku,
    _trafienia,
    TrackableType.PlaneWithinPolygon // tylko wewnątrz wykrytej płaszczyzny
);

if (!trafiono || _trafienia.Count == 0) return;

// Pierwszy wynik to najbliższy punkt trafienia
Pose pozaAR = _trafienia[0].pose;

if (_instancjaObiektu == null) {
    // Pierwsze umieszczenie obiektu
    _instancjaObiektu = Instantiate(prefabObiektu, pozaAR.position, pozaAR.ro-
tation);

    // Ukrycie płaszczyzn po umieszczeniu
    planeManager.requestedDetectionMode = PlaneDetectionMode.None;
    foreach (var p in planeManager.trackables)
        p.gameObject.SetActive(false);
}
else {
    // Przeniesienie już istniejącego obiektu
    _instancjaObiektu.transform.SetPositionAndRotation(
        pozaAR.position, pozaAR.rotation);
}
}
}

```

5.2. Obrót obiektu w stronę użytkownika

Umieszczony obiekt powinien być zwrócony w stronę użytkownika, a nie w losowym kierunku wynikającym z orientacji płaszczyzny. Osiąga się to przez modyfikację składowej Y obrotu.

Wymuszenie obrotu umieszczanego obiektu w kierunku użytkownika

C#

```

// Wymuszenie obrotu obiektu w stronę kamery (oś Y)
using UnityEngine;

Vector3 kierunekDoKamery = Camera.main.transform.position - pozaAR.position;
kierunekDoKamery.y = 0f; // Ignorujemy różnicę wysokości
Quaternion obrotDoKamery = Quaternion.LookRotation(kierunekDoKamery);

_instancjaObiektu.transform.SetPositionAndRotation(
    pozaAR.position,
    obrotDoKamery
);

```

5.3. Wskaźnik miejsca umieszczenia (placement indicator)

Przed umieszczeniem obiektu warto wyświetlić animowany wskaźnik pokazujący, w którym miejscu płaszczyzny pojawi się obiekt. Wskaźnik aktualizowany jest w każdej klatce, a promień AR jest wystrzelany ze środka ekranu zamiast z punktu dotknięcia.

Wskaźnik umieszczenia aktualizowany ze środka ekranu w każdej klatce

C#

```

void Update() {
    // Promień ze środka ekranu (nie z dotknięcia)
    Vector2 srodekEkranu = new Vector2(Screen.width / 2f, Screen.height / 2f);

    bool trafionyCenter = _raycastManager.Raycast(
        srodekEkranu,
        _trafienia,
        TrackableType.PlaneWithinPolygon
    );
    wskaznik.SetActive(trafionyCenter);

    if (trafionyCenter) {

```

```

        wskaznik.transform.SetPositionAndRotation(
            _trafienia[0].pose.position,
            _trafienia[0].pose.rotation
        );
    }
}

```

- Dokumentacja: [ARRaycastManager](#)

7. Śledzenie obrazów (image tracking)

Śledzenie obrazów to rodzaj AR opartego na znacznikach (marker AR). Aplikacja rozpoznaje wcześniej zdefiniowane obrazy (np. okładki książek, etykiety **produktów**, plakaty) w obrazie kamery i nakłada na nie wirtualną treść. Jest to technika mniej wymagająca obliczeniowo niż bez markerowy AR i działająca nawet na starszych urządzeniach.

6.1. Przygotowanie biblioteki referencyjnej obrazów

Zanim można będzie śledzić obrazy, należy stworzyć bibliotekę referencyjną (Reference Image Library) zawierającą wzorce do rozpoznania:

1. W oknie Project klikamy prawym przyciskiem myszy i wybieramy Create > XR > Reference Image Library.
2. W Inspektorze klikamy Add Image i przeciągamy plik graficzny (.png lub .jpg) z folderu projekt.
3. W polu Name wpisujemy nazwę identyfikującą obraz (będzie używana w skrypcie).
4. W polu Physical Size podajemy rzeczywiste wymiary obiektu w metrach (np. 0.1 x 0.1 dla znacznika 10 cm x 10 cm). Poprawny rozmiar jest niezbędny do trafnej estymacji odległości.
5. Zaznaczamy Specify Size i włączamy Keep Texture at Runtime.

UWAGA

Jako znaczników należy używać obrazów o wysokim kontraście i bogatej teksturze (dużo szczegółów, asymetryczny). Jednolite kolory, proste kształty geometryczne lub symetryczne wzory są źródłem błędów śledzenia. Narzędzie `arccoreimg.exe` (ARCore) lub XR Image Validation (Unity) pozwala ocenić jakość znacznika przed kompilacją.

6.2. Komponent AR Tracked Image Manager

AR Tracked Image Manager dodajemy do XR Origin. Konfigurujemy pola:

- Serialized Library: przypisujemy utworzoną bibliotekę referencyjną.
- Max Number of Moving Images: maksymalna liczba jednocześnie śledzonych obrazów. Zwiększenie wartości podnosi zużycie CPU.
- Tracked Image Prefab: opcjonalny prefabrykat tworzony automatycznie przy wykryciu obrazu.

6.3. Skrypt śledzenia obrazów

Zarządca śledzenia obrazów wyświetlający przypisane obiekty dla każdego znacznika

C#

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

```

```

/// <summary>
/// Wyświetla wirtualny obiekt przypisany do każdego śledzonego obrazu.
/// </summary>
[RequireComponent(typeof(ARTrackedImageManager))]
public class ZarzadcaObrazow : MonoBehaviour {
    [System.Serializable]
    public struct OdwzorowanieObrazu {
        public string nazwaObrazu;
        public GameObject prefabDoWyswietlenia;
    }

    [SerializeField] private OdwzorowanieObrazu[] odwzorowania;

    private ARTrackedImageManager _imageManager;
    private Dictionary<string, GameObject> _aktywneObiekty = new();

    void Awake() => _imageManager = GetComponent<ARTrackedImageManager>();

    void OnEnable() => _imageManager.trackedImagesChanged += NaZmianieObrazow;

    void OnDisable() => _imageManager.trackedImagesChanged -= NaZmianieObrazow;

    void NaZmianieObrazow(ARTrackedImagesChangedEventArgs args) {
        // Nowe śledzone obrazy
        foreach (var obraz in args.added)
            OdswiezObraz(obraz);

        // Zaktualizowane (zmiana pozycji lub stanu śledzenia)
        foreach (var obraz in args.updated)
            OdswiezObraz(obraz);

        // Utracone śledzenie
        foreach (var obraz in args.removed)
            UkryjObiektDlaObrazu(obraz.referenceImage.name);
    }

    void OdswiezObraz(ARTrackedImage obraz)
    {
        string nazwa = obraz.referenceImage.name;

        // Szukamy odpowiadającego prefabrykatu
        foreach (var odwz in odwzorowania) {
            if (odwz.nazwaObrazu != nazwa) continue;

            // Tworzenie instancji przy pierwszym wykryciu
            if (!_aktywneObiekty.ContainsKey(nazwa)) {
                var instancja = Instantiate(odwz.prefabDoWyswietlenia);
                _aktywneObiekty[nazwa] = instancja;
            }

            var obj = _aktywneObiekty[nazwa];

            // Śledzenie aktywne: pokaż i zaktualizuj pozycję
            bool aktywne = obraz.trackingState == TrackingState.Tracking;
            obj.SetActive(aktywne);

            if (aktywne) {
                obj.transform.SetPositionAndRotation(
                    obraz.transform.position,
                    obraz.transform.rotation
                );
            }
            break;
        }
    }

    void UkryjObiektDlaObrazu(string nazwa) {
        if (_aktywneObiekty.TryGetValue(nazwa, out var obj))
            obj.SetActive(false);
    }
}

```

WSKAZOWKA

TrackingState.Tracking oznacza aktywne śledzenie z pełną pewnością pozycji. TrackingState.Limited oznacza, że obraz jest widoczny, ale estymacja pozycji może być mniej dokładna. TrackingState.None oznacza utratę śledzenia - obraz nie jest widoczny w polu kamery.

- Dokumentacja: [ARTrackedImageManager](#)
- Dokumentacja: [arcoreimg - ocena jakości znacznika](#)

8. Śledzenie twarzy (face tracking)

Śledzenie twarzy to technologia rozpoznawania i śledzenia twarzy człowieka w obrazie kamery. AR Foundation obsługuje dwie konfiguracje: kamerę przednią (śledzenie twarzy skierowanej ku ekranowi, typowe dla filtrów) oraz kamerę tylną (rozpoznawanie twarzy osób w scenie). ARKit oferuje pełne wsparcie przez TrueDepth, ARCore ma ograniczone możliwości.

7.1. Konfiguracja kamery przedniej

Aby użyć śledzenia twarzy z kamerą przednią, należy zmienić konfigurację AR Camera Manager. W Inspektorze na komponencie AR Camera Manager ustawiamy Facing Direction na User (przednia). Następnie do XR Origin dodajemy komponent AR Face Manager.

Rys. 10. Komponent AR Face Manager z przypisanym prefabrykatem twarzy

7.2. Skrypt obsługi śledzenia twarzy

Zarządca śledzenia twarzy z obsługą dodawania, aktualizacji i usuwania

C#

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;

/// <summary>
/// Reaguje na wykrycie i utratę śledzenia twarzy.
/// Umieszcza wirtualną maskę na wykrytej twarzy.
/// </summary>
[RequireComponent(typeof(ARFaceManager))]
public class ZarzadcaTwarzy : MonoBehaviour {
    [SerializeField] private GameObject prefabMaski;

    private ARFaceManager _faceManager;

    void Awake() {
        _faceManager = GetComponent<ARFaceManager>();
    }
    void OnEnable() {
        _faceManager.facesChanged += NaZmianieTwarzy;
    }
    void OnDisable() {
        _faceManager.facesChanged -= NaZmianieTwarzy;
    }

    void NaZmianieTwarzy(ARFacesChangedEventArgs args)
    {
        foreach (var twarz in args.added) {
            Debug.Log($"Wykryto twarz: {twarz.trackableId}");
        }

        foreach (var twarz in args.updated) {
            // Pozycja i orientacja śledzonej twarzy
        }
    }
}
```

```

        Debug.Log($"Pożycja twarzy: {twarz.transform.position}");
    }

    foreach (var twarz in args.removed) {
        Debug.Log($"Utracono śledzenie twarzy: {twarz.trackableId}");
    }
}
}

```

7.3. Mieszanie kształtów twarzy (Blend Shapes) na iOS

ARKit na iPhone z kamerą TrueDepth udostępnia 52 parametry mieszania kształtów (blend shapes) opisujące wyraz twarzy: ruchy ust, powiek, brwi i policzków. Są one dostępne przez komponent AR Face z pakietu ARKit XR Plugin.

Odczyt współczynników blend shapes ARKit do animacji awatara	C#
<pre> #if UNITY_IOS using Unity.XR.CoreUtils; using UnityEngine.XR.ARMKit; using UnityEngine.XR.ARFoundation; /// <summary> /// Odczytuje współczynniki blend shapes twarzy z ARKit i steruje /// mięśniami awatara. /// </summary> public class MieszaczKształtow : MonoBehaviour { [SerializeField] private ARFace twarz; [SerializeField] private SkinnedMeshRenderer siatkaAwatara; void Update() { if (!ARKitFaceSubsystem.TryGetFaceCoefficients(twarz.trackableId, out var współczynniki)) return; // Przykład: przeniesienie ruchu ust do awatara if (współczynniki.TryGetValue(ARKitBlendShapeLocation.JawOpen, out float szczeka)) { // Indeks 0 = blend shape "Jaw_Open" na siatce awatara siatkaAwatara.SetBlendShapeWeight(0, szczeka * 100f); } } } #endif </pre>	

UWAGA

Śledzenie twarzy wymaga dostępu do kamery. Na iOS każde wymaganie dostępu do kamery (w tym kamera przednia) musi być opisane w polu Camera Usage Description w Player Settings. Brak opisu powoduje natychmiastowe odrzucenie aplikacji przez App Store.

- Dokumentacja: [ARFaceManager](#)
- Dokumentacja: [ARKit Blend Shapes](#)

9. Zakotwiczenie przestrzenne (AR Anchors)

Zakotwica AR (AR Anchor) to punkt w przestrzeni świata rzeczywistego trwale śledzony przez system AR. Obiekty Unity przypinane do zakotwicy utrzymują swoje położenie niezależnie od ruchu kamery. W odniesieniu do obiektów umieszczanych bezpośrednio w przestrzeni sceny, zakotwice zapewniają wyższą stabilność pozycji, szczególnie po długich sesjach AR lub przy rozległych scenach.

8.1. Komponent AR Anchor Manager

AR Anchor Manager dodajemy do XR Origin. Udostępnia metody tworzenia zakotwic (tak, będziemy używać takiego słowa) w dowolnym punkcie przestrzeni lub na wykrytej płaszczyźnie.

ZarządcaZakotwic.cs - Tworzenie zakotwicy AR na płaszczyźnie i przypinanie wirtualnego obiektu C#

```
using System.Collections.Generic;
using Unity.Physics;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

// Tworzy zakotwicę AR w miejscu dotknięcia i przypina do niej wirtualny obiekt.

[RequireComponent(typeof(ARRaycastManager))]
[RequireComponent(typeof(ARAnchorManager))]
public class ZarządcaZakotwic : MonoBehaviour {
    [SerializeField] private GameObject prefabObiektu;

    private ARRaycastManager _raycastManager;
    private ARAnchorManager _anchorManager;
    private List<ARRaycastHit> _trafienia = new();
    private List<ARAnchor> _zakotwice = new();

    void Awake() {
        _raycastManager = GetComponent<ARRaycastManager>();
        _anchorManager = GetComponent<ARAnchorManager>();
    }

    // Tworzy zakotwicę w punkcie wskazanym dotknięciem.
    // Wywoływana z przycisku UI przez OnClick().

    public void UtworzZakotwice(Vector2 pozycjaEkranu) {
        bool trafiono = _raycastManager.Raycast(
            pozycjaEkranu, _trafienia, TrackableType.PlaneWithinPolygon);

        if (!trafiono) return;
        Pose poza = _trafienia[0].pose;

        // Tworzenie zakotwicy w znalezionym punkcie
        ARAnchor zakotwica = _anchorManager.AttachAnchor(
            (ARPlane)_trafienia[0].trackable,
            poza
        );
        if (zakotwica == null) {
            Debug.LogWarning("Nie udało się utworzyć zakotwicy.");
            return;
        }

        // Przypinanie obiektu do zakotwicy jako dziecka
        var obiekt = Instantiate(prefabObiektu,
            zakotwica.transform.position,
            zakotwica.transform.rotation);
        obiekt.transform.SetParent(zakotwica.transform, worldPositionStays: true);

        _zakotwice.Add(zakotwica);
        Debug.Log($"Zakotwica utworzona: {zakotwica.trackableId}");
    }

    // Usuwa wszystkie zakotwice i przypięte obiekty.

    public void UsunWszystkieZakotwice() {
        foreach (var zakotwica in _zakotwice)
        {
            if (zakotwica != null)
                Destroy(zakotwica.gameObject);
        }
    }
}
```

```

    _zakotwice.Clear();
}

```

8.2. Trwałość zakotwiczeń między sesjami (cloud anchors)

Standardowe zakotwice AR istnieją tylko przez czas trwania sesji. Gdy użytkownik zamknie aplikację i otworzy ją ponownie, zakotwice są utracone. ARCore Cloud Anchors (Google) oraz ARKit Persistent World Anchors (Apple) pozwalają zapisać zakotwice na serwerze i przywrócić je w nowej sesji, a nawet udostępnić innym użytkownikom.

INFORMACJA

Integracja Cloud Anchors wymaga konfiguracji projektu w Google Cloud Console (ARCore API) lub App Store Connect (ARKit). Szczegółowy opis wykracza poza program tego laboratorium. Zainteresowanych odsyłamy do dokumentacji ARCore Persistent Cloud Anchors i [Zadania dodatkowego 4](#).

- Dokumentacja: [ARAnchorManager](#)
- Dokumentacja: [ARCore Cloud Anchors - dokumentacja](#)

10. Optymalizacja i testowanie aplikacji AR

Aplikacje AR są szczególnie wymagające pod względem wydajności: procesor obsługuje jednocześnie śledzenie świata, dekodowanie obrazu z kamery oraz główną pętlę gry. Temperatura urządzenia szybko rośnie, co może prowadzić do ograniczenia taktowania (*thermal throttling*) i spadku wydajności. Poniżej zebrano kluczowe techniki optymalizacji specyficzne dla AR.

9.1. Ograniczanie liczby śledzonych obiektów

Komponent	Parametr	Zalecenie
AR Plane Manager	requestedDetectionMode	Ustaw None po umieszczeniu obiektu
AR Tracked Image Mgr	maxNumberOfMovingImages	Ustaw 1 dla prostych aplikacji (max 4)
AR Face Manager	maximumFaceCount	Ustaw 1 (jedno śledzenie twarzy)
AR Anchor Manager	liczba zakotwic	Unikaj setek zakotwic w jednej sesji

9.2. Optymalizacja renderowania

Poza ogólnymi zasadami z Laboratorium Unity 3 (Draw Call Batching, kompresja ASTC, Object Pooling) w aplikacjach AR obowiązują dodatkowe zasady:

- Wyłącz cieniowanie dynamiczne (Shadows: None) dla większości obiektów AR. Cienie rzucane na rzeczywistą podłogę można realizować przez płaską siatkę z teksturą błysku (blob shadow).
- Użyj shaderów zoptymalizowanych dla AR (Mobile > Unlit lub Universal Render Pipeline > Lit z minimalną liczbą próbek światła).
- Ograniczaj liczbę polygonów modeli AR do 5000-20 000 trójkątów dla typowych obiektów. Modele 3D powinny mieć poziomy szczegółowości (LOD).
- Włącz GPU Instancing dla powtórzonych obiektów (np. drzew, znaczników). Zmniejsza liczbę Draw Calls przy wielu instancjach tego samego mesh-a.

9.3. Testowanie w środowisku XR Simulation

AR Foundation 5.x wprowadza środowisko XR Simulation, które pozwala testować aplikacje AR bezpośrednio w edytorze Unity bez fizycznego urządzenia. W oknie Simulation Environment (Window > XR > AR Foundation > XR Environment) wybieramy gotowe sceny testowe lub tworzymy własne.

Rys. 14. Okno XR Environment z wirtualną sceną testową dla AR Foundation

Komentarz dokumentujący możliwości i ograniczenia XR Simulation	C#
<pre>// Plik: Assets/XR/Settings/XRSimulationRuntimeSettings.asset // Domyślnie XR Simulation symuluje: // - Wykrywanie płaszczyzn poziomych i pionowych // - Śledzenie obrazów (z biblioteką referencyjną) // - Estymację oświetlenia // NIE symuluje: // - Śledzenia twarzy (wymaga fizycznego urządzenia) // - Cloud Anchors // - Siatek świata (World Meshes)</pre>	

9.4. Profiler dla AR

Podczas profilowania na fizycznym urządzeniu zwracamy uwagę na następujące sygnatury w Unity Profiler:

Sygnatura w Profilerze	Możliwa przyczyna
ARSession.Update > 8 ms	Zbyt wiele aktywnych podsystemów AR jednocześnie
Camera.Render > 16 ms	Za dużo polygonów lub zbyt złożone shadery
GC.Collect > 1 ms co klatkę	Alokacje w Update() (LINQ, new w pętli, string concat)
Physics.Simulate > 4 ms	Zbyt mały Fixed Timestep lub za dużo Colliderów
Thermal Throttling	Temperatura procesora: zmniejsz renderowane FPS do 30

WSKAZOWKA

Ustaw `Application.targetFrameRate = 30` dla aplikacji AR. Limit 60 FPS jest rzadko osiągalny w AR bez nagrzewania urządzenia. 30 FPS w AR jest postrzegane przez użytkowników jako płynne i znacznie wydłuża czas pracy na baterii.

- Dokumentacja: [XR Simulation](#)
- Dokumentacja: [AR Foundation - optymalizacja](#)

11. Zadania dla studenta

Poniższe zadania należy wykonać samodzielnie, tworząc nowy projekt Unity z konfiguracją AR Foundation. Zadania 1-4 można przetestować w środowisku XR Simulation w edytorze. Zadania 5 i 6 wymagają fizycznego urządzenia z systemem Android lub iOS.

1. Konfiguracja projektu AR: załóż nowy projekt Unity 6 z szablonem Universal Render Pipeline. Zainstaluj pakiety AR Foundation 6.x, ARCore XR Plugin i ARKit XR Plugin. Skonfiguruj XR Plug-in Management dla Androida i iOS. Dodaj do sceny obiekty AR Session i XR

Origin. Przetestuj działanie w środowisku XR Simulation - upewnij się, że płaszczyzny są wykrywane poprawnie.

2. Tap-to-place: zaimplementuj umieszczanie wybranego modelu 3D (np. cube, sfera lub model pobrany z Unity Asset Store) w miejscu dotknięcia na wykrytej płaszczyźnie poziomej. Obiekt powinien być zwrócony w stronę użytkownika oraz posiadać animowany wskaźnik miejsca umieszczenia widoczny przed dotknięciem. Po umieszczeniu obiektu wyłącz dalsze wykrywanie płaszczyzn i ukryj ich wizualizację.
3. Śledzenie obrazów: przygotuj bibliotekę referencyjną z co najmniej dwiema różnymi grafikami (np. logo uczelni, okładka książki). Dla każdego znacznika przypisz inny wirtualny obiekt 3D. Obiekty powinny pojawiać się dokładnie na wykrytym obrazie i znikać po wyjściu poza pole widzenia kamery. Zweryfikuj jakość znaczników narzędziem `arcoreimg`.
4. Estymacja oświetlenia: zaimplementuj dynamiczne dostosowanie oświetlenia sceny Unity do warunków rzeczywistych. Użyj danych z `ARCameraFrameEventArgs` (`averageBrightness`, `colorCorrection`) do aktualizacji `RenderSettings.ambientIntensity` i `RenderSettings.ambientLight` w każdej klatce. Porównaj wygląd obiektów z i bez estymacji oświetlenia.
5. Zakotwiczanie obiektów: zaimplementuj aplikację pozwalającą umieszczać wiele obiektów w przestrzeni z zakotwiczaniem każdego do płaszczyzny. Każdy obiekt powinien utrzymywać stabilną pozycję podczas poruszania kamerą. Dodaj przycisk Usuń wszystkie usuwający wszystkie zakotwice i obiekty ze sceny.
6. Zbuduj i wgraj aplikację AR na fizyczne urządzenie Android. Przetestuj działanie wykrywania płaszczyzn, tap-to-place i śledzenia obrazów na urządzeniu. Zmierz średnią liczbę klatek na sekundę (FPS) i liczbę Draw Calls przez Unity Profiler przy połączeniu USB. Zanotuj wyniki i porównaj z wartościami uzyskanymi w środowisku XR Simulation.

12. Zadania dodatkowe

Zadania dodatkowe wymagają samodzielnego przeszukania dokumentacji i **zasobów** zewnętrznych. Przeznaczone są dla studentów dążących do głębszego zrozumienia technologii AR i mogą być uwzględnione przy ocenie aktywności laboratoryjnej.

- Filtr twarzy: zaimplementuj prosty filtr twarzy dla kamery przedniej. W miejscu nosa, ust lub oczu umieść wirtualne elementy graficzne (np. okulary, wąsy, kapelusz) korzystając z komponentu AR Face Manager. Elementy powinny śledzić ruch twarzy i dostosować skalę do odległości kamery. Na iOS przetestuj również odczyt blend shapes dla otwierania ust i mrużenia oczu.
- Siatki świata z LiDAR (iOS): na urządzeniu z LiDAR (iPad Pro 2020+, iPhone 12 Pro+) włącz komponent AR Mesh Manager i zwizualizuj siatkę świata. Skonfiguruj okluzję przez głębość (Occlusion Manager) tak, aby wirtualne obiekty były poprawnie przesłaniane przez rzeczywiste meble i ściany. Przetestuj różnicę jakości okluzji z i bez LiDAR.
- Wieloosobowe AR z Netcode for GameObjects: zaimplementuj prostą aplikację wieloosobową, w której dwaj użytkownicy widzą ten sam obiekt w przestrzeni rzeczywistej. Użyj pakietu Netcode for GameObjects (Unity Gaming Services) do synchronizacji pozycji obiektów przez sieć lokalną. Punktem referencyjnym dla obu urządzeń niech będzie wspólny znacznik AR (śledzenie obrazów).

- Cloud Anchors: skonfiguruj projekt w Google Cloud Console (ARCore API) i zaimplementuj zapis zakotwicy do chmury. Po zapisaniu pokaż użytkownikowi kod identyfikatora zakotwicy. Dodaj ekran wczytywania, na którym użytkownik wpisuje kod i przywróci zapisaną zakotwicę w nowej sesji, umieszczając obiekt w dokładnie tym samym miejscu.
- Wideo AR: zamiast statycznego modelu 3D wyświetl film wideo na wykrytym obrazie-znaczniku. Użyj komponentu Video Player z Unity i ustaw RenderTexture jako teksturę obiektu Quad przypiętego do śledzonego obrazu. Film powinien startować automatycznie po wykryciu znacznika i zatrzymywać się po wyjściu poza kadr kamery.

Przydatne zasoby do zadań dodatkowych:

- Dokumentacja: [AR Foundation Samples - GitHub](#)
- Dokumentacja: [ARCore Cloud Anchors](#)
- Dokumentacja: [Netcode for GameObjects - pierwsze kroki](#)
- Dokumentacja: [ARKit Mesh Manager - dokumentacja](#)
- Dokumentacja: [Unity Video Player](#)

ĆWICZENIE LABORATORYJNE - UNITY 6

Sceny · PlayerPrefs · Animacje · Audio

Programowanie Aplikacji Mobilnych - moduł U

1. Spis treści

1. Spis treści	85
2. Wprowadzenie i założenia projektu gry	85
3. Architektura projektu i ScriptableObjects	86
4. Konfiguracja środowiska AR z Laboratorium Unity 5	89
5. System czasu rzeczywistego i kalendarza	90
6. Mechaniki gry oparte na czasie doby i porze roku	93
7. Sterowanie przez sensory IMU: akcelerometr i żyroskop	96
8. Dynamiczne obiekty AR reagujące na ruch urządzenia	99
9. System nagród oparty na kalendarzu i zdarzeniach specjalnych	101
10. Interfejs użytkownika i HUD gry	103
11. Budowanie i testowanie gry na urządzeniu mobilnym	105

2. Wprowadzenie i założenia projektu gry

Niniejsze laboratorium stanowi kulminację całego cyklu. Łączy elementy omawiane na poprzednich pięciu zajęciach: interfejs użytkownika z Unity UI, nawigację między scenami, obsługę dotyku i sensorów IMU, komunikację sieciową, architekturę danych oraz rzeczywistość rozszerzoną. Celem jest stworzenie kompletnej, grywalnej gry mobilnej AR, w której mechaniki zależą od rzeczywistego czasu, daty i ruchu urządzenia.

1.1. Opis gry: AR Garden

PROJEKT GRY

AR Garden to gra AR, w której gracz pielęgnuje wirtualny ogród umieszczony na rzeczywistej powierzchni. Rośliny rosną w tempie uzależnionym od pory dnia i pory roku (odczytywanych z zegara systemowego). Podlewanie realizowane jest przez potrząśnięcie urządzenia (akcelerometr). Obrót urządzenia zmienia kąt padania wirtualnego słońca (żyroskop). Kalendarzowe eventy (urodziny, święta) odblokowują specjalne rośliny i nagrody. Gracz zbiera punkty za codzienną pielęgnację.

1.2. Mechaniki gry i powiązanie z danymi systemowymi

Mechanika gry	Dane systemowe	Klasa / API Unity
Wzrost roślin	Godzina doby, pora roku	System.DateTime
Podlewanie	Potrząśnięcie urządzenia	Input.acceleration
Kąt słońca	Orientacja urządzenia	Input.gyro / AttitudeSensor
Warunki pogodowe	Miesiąc roku	System.DateTime.Month
Specjalne rośliny	Data kalendarza	System.DateTime.Day/Month
Seria dzienna (streak)	Data ostatniego logowania	PlayerPrefs + DateTime
Nocny tryb wizualny	Godzina doby	RenderSettings, Light

1.3. Struktura projektów i scen

Projekt składa się z trzech scen.

Struktura folderów projektu AR Garden	
Sceny projektu AR Garden	
BootScene	<-- Inicjalizacja: sprawdzenie AR, zapis/odczyt danych, przejście
MenuScene	<-- Menu główne: statystyki, sklep roślin, ustawienia
GameScene	<-- Scena główna AR z ogrodem i HUD
Struktura folderów Assets/	
Scripts/	
Core/	<-- GameManager, TimeManager, CalendarManager
AR/	<-- ARGardenManager, PlantPlacer, PlantController
Sensors/	<-- IMUSensorManager, ShakeDetector, TiltController
UI/	<-- HUDController, MenuController, NotificationUI
Data/	<-- ScriptableObjects: PlantSO, SeasonEventSO, RewardSO
Prefabs/	
Plants/	<-- Prefabrykaty roślin (wiosna, lato, jesień, zima)
FX/	<-- Efekty cząsteczkowe (deszcz, śnieg, potrząśnięcie)
ScriptableObjects/	
Plants/	<-- Definicje roślin (.asset)
Events/	<-- Zdarzenia kalendarzowe (.asset)

3. Architektura projektu i ScriptableObjects

Projekt stosuje architekturę opartą na danych (Data-Driven Design) z użyciem ScriptableObjects jako głównego repozytorium konfiguracji. Menedżerzy (Managerowie) są wzorcem Singleton z DontDestroyOnLoad. Komunikacja między komponentami odbywa się przez assetowe kanały zdarzeń (SO Event Channels) opisane w Laboratorium Unity 4.

2.1. ScriptableObject rośliny (PlantSO)

ScriptableObject definiujący właściwości rośliny w grze AR Garden

```
using UnityEngine;

[CreateAssetMenu(fileName = "NowaRoslina",
                  menuName = "AR Garden/Definicja Rosliny")]
public class PlantSO : ScriptableObject
{
    [Header("Informacje podstawowe")]
    public string nazwaRosliny;
    public string opis;
    public Sprite ikonaUI;
    public GameObject prefabAR;           // prefabrykat 3D w scenie AR

    [Header("Warunki wzrostu")]
    public PoraRoku[] preferowanePory;   // pory roku sprzyjające wzrostowi
    public Vector2 godzinyWzrostu;       // np. (6, 20) = wzrost od 6:00 do 20:00
    public float wspolczynnikWzrostu;    // jednostki wzrostu na minutę
    public float maksymalnaWysokosc;     // metry w przestrzeni AR

    [Header("Podlewanie")]
    public float interwaPodlewania;      // godziny między podlewaniami
    public float bonusWzrostuPoPodle;    // mnożnik wzrostu po podlaniu

    [Header("Odblokowywanie")]
    public bool specjalna;                // dostępna tylko przez event kalendarzowy
    public int wymaganyPoziom;           // minimalny poziom gracza
    public int cenaZiaren;               // waluta w grze

    [Header("Wizualizacja")]
    public Gradient kolorLisci;           // zmiana koloru z poryą roku
    public float progKwitnienia;         // 0..1, przy którym pojawia się kwiat
}

[System.Serializable]
public enum PoraRoku { Wiosna, Lato, Jesien, Zima }
```

2.2. ScriptableObject zdarzenia kalendarzowego (CalendarEventSO)

ScriptableObject opisujący zdarzenie kalendarzowe z nagrodą

```
using UnityEngine;
using System;

[CreateAssetMenu(fileName = "NoweZdarzenie",
                  menuName = "AR Garden/Zdarzenie Kalendarzowe")]
public class CalendarEventSO : ScriptableObject
{
    [Header("Termin")]
    public int dzien;                    // 1-31
    public int miesiac;                  // 1-12 (0 = każdy miesiąc)
    public bool doroczne;                // czy zdarzenie powtarza się co roku

    [Header("Treść zdarzenia")]
    public string tytulZdarzenia;
    public string opisZdarzenia;
    public Sprite baner;

    [Header("Nagroda")]
    public PlantSO odblokowanaroslina;  // null = brak nowej rośliny
    public int bonusZiaren;
    public GameObject efektSpecjalny;    // prefabrykat AR (np. fajerwerki)

    /// <summary>
    /// Sprawdza, czy zdarzenie jest aktywne dla podanej daty.
    /// </summary>
    public bool CzyAktywne(DateTime data)
```

```

    {
        bool pasujeDzien    = dzien    == 0 || data.Day    == dzien;
        bool pasujeMiesiac  = miesiac  == 0 || data.Month  == miesiac;
        return pasujeDzien && pasujeMiesiac;
    }
}

```

2.3. Menedżer gry (GameManager)

GameManager to globalny Singleton przechodzący między scenami. Przechowuje stan gry, koordynuje menedżerów i udostępnia im referencje przez właściwości statyczne.

GameManager jako Singleton trwający między scenami

```

using UnityEngine;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }

    // Referencje do menedżerów
    public static TimeManager  Czas      { get; private set; }
    public static CalendarManager Kalendarz { get; private set; }
    public static IMUSensorManager Sensory { get; private set; }

    // Stan gry
    public int    Poziom      { get; private set; }
    public int    Ziarna      { get; private set; }
    public int    SerialDni    { get; private set; }

    void Awake()
    {
        if (Instance != null) { Destroy(gameObject); return; }
        Instance = this;
        DontDestroyOnLoad(gameObject);

        // Inicjalizacja menedżerów
        Czas      = GetComponentInChildren<TimeManager>();
        Kalendarz = GetComponentInChildren<CalendarManager>();
        Sensory    = GetComponentInChildren<IMUSensorManager>();

        WczytajStan();
    }

    void OnApplicationPause(bool pauza)
    {
        if (pauza) ZapiszStan();
    }

    void OnApplicationQuit()
    {
        ZapiszStan();
    }

    public void DodajZiarna(int ilosc)
    {
        Ziarna += ilosc;
        PlayerPrefs.SetInt("Ziarna", Ziarna);
    }

    void WczytajStan()
    {
        Poziom    = PlayerPrefs.GetInt("Poziom", 1);
        Ziarna    = PlayerPrefs.GetInt("Ziarna", 50);
        SerialDni = PlayerPrefs.GetInt("Seria", 0);
    }

    void ZapiszStan()
    {

```

```

        PlayerPrefs.SetInt("Poziom", Poziom);
        PlayerPrefs.SetInt("Ziarna", Ziarna);
        PlayerPrefs.SetInt("Seria", SeriaDni);
        PlayerPrefs.Save();
    }
}

```

- Dokumentacja: [ScriptableObject - dokumentacja](#)

4. Konfiguracja środowiska AR z Laboratorium Unity 5

Projekt AR Garden korzysta z wszystkich elementów AR Foundation opisanych w Laboratorium Unity 5. Poniżej zestawiono minimalną konfigurację wymaganą do uruchomienia gry, uzupełniając ją o elementy specyficzne dla środowiska ogrodowego.

3.1. Hierarchia sceny GameScene

Hierarchia obiektów w głównej scenie gry AR Garden

```

GameScene hierarchia:
  GameManager      <-- Singleton z podrzednymi menedżerami
    TimeManager
    CalendarManager
    IMUSensorManager
    AudioManager

  AR Session
    AR Input Manager

  XR Origin
    Camera Offset
      Main Camera    <-- AR Camera Manager + AR Camera Background
    AR Plane Manager <-- PlaneDetectionMode: HorizontalUpward
    AR Raycast Manager
    AR Anchor Manager
    ARGardenManager  <-- Skrypt zarządzający ogółem (sekcja 7)

  Canvas (Screen Space Overlay)
    SafeAreaContainer <-- SafeAreaFitter (Lab 3)
    HUD               <-- Czas, ziarna, pasek podlewania
    PanelRosliny      <-- Lista roślin do zasadzenia
    PanelZdarzenia    <-- Pop-up zdarzenia kalendarzowego

```

3.2. Konfiguracja oświetlenia AR dla pory dnia

Gra dostosowuje oświetlenie sceny do godziny rzeczywistej. Kierunkowe światło globalne (Directional Light) symuluje słońce, którego kąt padania i intensywność zmieniają się w zależności od pory dnia. Komponent AR Camera Background dba o poprawne wyświetlanie obrazu kamery pod obiektami 3D:

Skrypt słońca AR dostosowujący oświetlenie do rzeczywistej pory dnia

```

using UnityEngine;

/// <summary>
/// Dostosowuje oświetlenie sceny AR do rzeczywistej pory dnia.
/// Skrypt przypisujemy do obiektu Directional Light w scenie.
/// </summary>
public class SlonceAR : MonoBehaviour
{
    [Header("Parametry słońca")]
    [SerializeField] private Gradient kolorSwiatla;
    [SerializeField] private AnimationCurve intensywnosc;
    [SerializeField] private AnimationCurve katPionowy; // 0=polnoc, 0.5=poludnie, 1=polnoc

    [Header("Referencje")]

```

```

[SerializeField] private Light swiatloKierunkowe;

void Update()
{
    // Czas doby jako wartosc 0..1 (0 = 00:00, 0.5 = 12:00, 1 = 24:00)
    float tDoby = (float)System.DateTime.Now.TimeOfDay.TotalHours / 24f;

    // Aktualizacja koloru, intensywnosci i kata swiatla
    swiatloKierunkowe.color = kolorSwiatla.Evaluate(tDoby);
    swiatloKierunkowe.intensity = intensywnosc.Evaluate(tDoby);

    float katX = katPionowy.Evaluate(tDoby) * 180f - 90f;
    swiatloKierunkowe.transform.rotation = Quaternion.Euler(katX, -30f, 0f);

    // Dostosowanie swiatla otoczenia do nocy
    float jasnoc = Mathf.Clamp01(swiatloKierunkowe.intensity);
    RenderSettings.ambientIntensity = Mathf.Lerp(0.05f, 1f, jasnoc);
}

```

WSKAZOWKA

Gradient i AnimationCurve konfigurowane są w Inspektorze bez modyfikacji kodu. Pozwala to projektantowi gry dostosować wygląd oświetlenia bez ingerencji w skrypty. Typowe punkty kluczowe krzywej intensywności: 0.0 (północ) = 0.05, 0.25 (wschód) = 0.3, 0.5 (południe) = 1.0, 0.75 (zachód) = 0.3, 1.0 (północ) = 0.05.

5. System czasu rzeczywistego i kalendarza

Klasa System.DateTime w języku C# udostępnia pełen dostęp do daty i czasu systemowego. W grze AR Garden stanowi ona podstawę każdej mechaniki zależnej od czasu: wzrostu roślin, warunków pogodowych, zdarzeń specjalnych i serii dziennych.

4.1. Klasa TimeManager

TimeManager - centralny menedżer czasu z zdarzeniami pory dnia i nowego dnia

```

using UnityEngine;
using System;
using UnityEngine.Events;

/// <summary>
/// Centralny menedżer czasu i pory dnia w grze AR Garden.
/// Udostępnia wartości pochodne od DateTime.Now i wywołuje
/// zdarzenia przy zmianie pory dnia oraz o północy (nowy dzień).
/// </summary>
public class TimeManager : MonoBehaviour
{
    // Zdarzenia
    public static event Action<PoraDnia> NaZmianiePoryDnia;
    public static event Action<DateTime> NaNowyDzien;

    // Właściwości dostępne globalnie
    public static DateTime Teraz => DateTime.Now;
    public static PoraDnia AktualnaPora { get; private set; }
    public static PoraRoku AktualnaPoroku { get; private set; }
    public static float TDoby { get; private set; } // 0..1

    private PoraDnia _ostatniaPora;
    private DateTime _ostatniaDzien;

    void Start()
    {
        _ostatniaDzien = DateTime.Today;
    }
}

```

```

        OdswiezStan();
    }

    void Update()
    {
        OdswiezStan();
    }

    void OdswiezStan()
    {
        DateTime teraz = DateTime.Now;

        // Czas doby jako 0..1
        TDoby = (float)teraz.TimeOfDay.TotalHours / 24f;

        // Obliczenie pory dnia
        PoraDnia nowaPora = ObliczPoreDnia(teraz.Hour);
        if (nowaPora != _ostatniaPora)
        {
            _ostatniaPora = nowaPora;
            AktualnaPora = nowaPora;
            NaZmianiePoryDnia?.Invoke(nowaPora);
        }

        // Obliczenie pory roku
        AktualnaPoroku = ObliczPoraRoku(teraz.Month);

        // Wykrycie nowego dnia
        if (teraz.Date != _ostatniaDzien.Date)
        {
            _ostatniaDzien = teraz;
            NaNowyDzien?.Invoke(teraz);
        }
    }

    public static PoraDnia ObliczPoreDnia(int godzina)
    {
        if (godzina >= 5 && godzina < 9) return PoraDnia.Swit;
        if (godzina >= 9 && godzina < 17) return PoraDnia.Dzien;
        if (godzina >= 17 && godzina < 21) return PoraDnia.Zmierzch;
        return PoraDnia.Noc;
    }

    public static PoraRoku ObliczPoraRoku(int miesiac)
    {
        return miesiac switch
        {
            3 or 4 or 5 => PoraRoku.Wiosna,
            6 or 7 or 8 => PoraRoku.Lato,
            9 or 10 or 11 => PoraRoku.Jesien,
            - => PoraRoku.Zima
        };
    }

    /// <summary>
    /// Oblicza, ile minut rzeczywistego czasu upłynęło od zapisu.
    /// Używane do obliczenia wzrostu roślin podczas nieobecności gracza.
    /// </summary>
    public static double MinutyOdZapisu(string klucz)
    {
        string zapisany = PlayerPrefs.GetString(klucz, string.Empty);
        if (string.IsNullOrEmpty(zapisany)) return 0;

        if (DateTime.TryParse(zapisany, out DateTime ostatniZapis))
            return (DateTime.Now - ostatniZapis).TotalMinutes;

        return 0;
    }

```

```

    public static void ZapiszTeraz(string klucz)
    {
        PlayerPrefs.SetString(klucz, DateTime.Now.ToString("O"));
        PlayerPrefs.Save();
    }
}

public enum PoraDnia { Swit, Dzień, Zmierzch, Noc }

```

4.2. Klasa CalendarManager

CalendarManager wykrywający aktywne zdarzenia kalendarzowe

```

using UnityEngine;
using System;
using System.Collections.Generic;

/// <summary>
/// Sprawdza aktywne zdarzenia kalendarzowe i wywołuje odpowiednie nagrody.
/// Lista zdarzeń konfigurowana jest jako tablica ScriptableObject w Inspektorze.
/// </summary>
public class CalendarManager : MonoBehaviour
{
    public static event Action<CalendarEventSO> NaZdarzenieKalendarzowe;

    [SerializeField] private CalendarEventSO[] wszystkieZdarzenia;

    private readonly List<CalendarEventSO> _aktywneToday = new();

    void Start()
    {
        TimeManager.NaNowyDzień += SprawdzZdarzenia;
        SprawdzZdarzenia(DateTime.Now);
    }

    void OnDestroy()
    {
        TimeManager.NaNowyDzień -= SprawdzZdarzenia;
    }

    void SprawdzZdarzenia(DateTime data)
    {
        _aktywneToday.Clear();

        foreach (var zdarzenie in wszystkieZdarzenia)
        {
            if (zdarzenie.CzyAktywne(data))
            {
                _aktywneToday.Add(zdarzenie);
                NaZdarzenieKalendarzowe?.Invoke(zdarzenie);
                Debug.Log($"Aktywne zdarzenie: {zdarzenie.tytułZdarzenia}");
            }
        }
    }

    /// <summary>
    /// Zwraca listę aktywnych dziś zdarzeń kalendarzowych.
    /// </summary>
    public IReadOnlyList<CalendarEventSO> AktywneZdarzenia => _aktywneToday;
}

```

4.3. Seria dzienna (daily streak)

Seria dzienna to licznik kolejnych dni, w których gracz uruchomił aplikację. Przerywa się, gdy gracz pomija dzień. Wartość serii wpływa na nagrody i mnożnik zdobywanych ziaren:

Logika serii dziennych logowań oparta na DateTime

```

/// <summary>
/// Sprawdza serię dziennych logowań i aktualizuje licznik.
/// Wywoływać raz przy każdym starcie aplikacji.
/// </summary>
public static void AktualizujSerie()
{
    const string KLUCZ_OSTATNI = "OstatnieLogowanie";
    const string KLUCZ_SERIA = "Seria";

    string ostatni = PlayerPrefs.GetString(KLUCZ_OSTATNI, string.Empty);
    int seria = PlayerPrefs.GetInt(KLUCZ_SERIA, 0);

    if (!string.IsNullOrEmpty(ostatni) &&
        DateTime.TryParse(ostatni, out DateTime ostatnieDzien))
    {
        double roznica = (DateTime.Today - ostatnieDzien.Date).TotalDays;

        if (roznica < 1)
        {
            // To samo uruchomienie tego samego dnia - seria bez zmian
            Debug.Log($"Ten sam dzień. Seria: {seria}");
            return;
        }
        else if (roznica == 1)
        {
            // Kolejny dzień z rzędu - seria rośnie
            seria++;
            Debug.Log($"Seria przedłużona! Dzień {seria}.");
        }
        else
        {
            // Przerwa większa niż 1 dzień - seria zerowana
            seria = 1;
            Debug.Log("Seria przerwana. Zaczynam od nowa.");
        }
    }
    else
    {
        // Pierwsze uruchomienie
        seria = 1;
    }

    PlayerPrefs.SetInt(KLUCZ_SERIA, seria);
    PlayerPrefs.SetString(KLUCZ_OSTATNI, DateTime.Today.ToString("O"));
    PlayerPrefs.Save();
}

```

6. Mechaniki gry oparte na czasie doby i porze roku

Czas rzeczywisty decyduje o tym, jak zachowuje się świat gry. Rośliny rosną szybciej w cieniu dnia i spowalniają nocą. Niektóre gatunki rozkwitają tylko latem, inne przetrwają zimę. Mechaniki te nie wymagają od gracza czekania przy ekranie - system oblicza postęp również dla czasu spędzonego poza grą.

5.1. Kontroler wzrostu rośliny (PlantController)

PlantController - pełna logika wzrostu, wzrost offline i podlewanie

```

using UnityEngine;
using System;
using System.Linq;

/// <summary>
/// Kontroluje wzrost, podlewanie i wizualizację pojedynczej rośliny AR.

```

```

/// Każda roślina jest instancją tego skryptu z własnym PlantSO.
/// </summary>
public class PlantController : MonoBehaviour
{
    [SerializeField] private PlantSO      definicja;
    [SerializeField] private Transform    model3D;
    [SerializeField] private ParticleSystem efektKwitnienia;

    // Stan trwały (zapisywany przez ID rośliny)
    private string KluczWysokosc => "Wys_" + definicja.nazwaRosliny;
    private string KluczPodlew  => "Pod_" + definicja.nazwaRosliny;

    private float _wysokosc;      // aktualna wysokość (0..max)
    private bool  _kwitnie;

    void Start()
    {
        // Wczytanie zapisanego stanu
        _wysokosc = PlayerPrefs.GetFloat(KluczWysokosc, 0f);

        // Doliczenie wzrostu podczas nieobecności gracza
        double minutyPrzerwy = TimeManager.MinutyOdZapisu("OstatniZapis");
        if (minutyPrzerwy > 0)
            DoliczonWzrost((float)minutyPrzerwy);

        OdswiezModel();
    }

    void Update()
    {
        // Wzrost w czasie rzeczywistym
        float przyrost = ObliczPrzyrost(Time.deltaTime / 60f);
        _wysokosc = Mathf.Clamp(_wysokosc + przyrost, 0f, definicja.maksymalnaWysokosc);
        OdswiezModel();
    }

    void OnDestroy()
    {
        PlayerPrefs.SetFloat(KluczWysokosc, _wysokosc);
    }

    /// <summary>
    /// Oblicza przyrost wysokości (metry) za podaną liczbę minut.
    /// Uwzględnia porę dnia i porę roku.
    /// </summary>
    float ObliczPrzyrost(float minuty)
    {
        // Baza: współczynnik z definicji rośliny
        float baza = definicja.wspolczynnikWzrostu * minuty;

        // Mnożnik pory dnia
        float mnoznikPoryDnia = TimeManager.AktualnaPora switch
        {
            PoraDnia.Swit      => 0.8f,
            PoraDnia.Dzien     => 1.0f,
            PoraDnia.Zmierzch  => 0.6f,
            PoraDnia.Noc       => 0.2f,
            _                   => 1.0f
        };

        // Mnożnik pory roku (czy roślina preferuje tą porę)
        bool preferowana = definicja.preferowanePory.Contains(TimeManager.AktualnaPoroku);
        float mnoznikPoryRoku = preferowana ? 1.5f : 0.5f;

        // Sprawdzenie zakresu godzin wzrostu
        float h = (float)DateTime.Now.TimeOfDay.TotalHours;
        bool wGodzinach = h >= definicja.godzinyWzrostu.x &&
            h <= definicja.godzinyWzrostu.y;
        float mnoznikGodzin = wGodzinach ? 1f : 0.1f;
    }
}

```

```

        return baza * mnoznikPoryDnia * mnoznikPoryRoku * mnoznikGodzin;
    }

    /// <summary>
    /// Dolicza wzrost za czas nieobecności gracza w uproszczeniu.
    /// Przyjmuje średni mnożnik pory dnia 0.5 dla całej przerwy.
    /// </summary>
    void DoliczonWzrost(float minuty)
    {
        float przyrost = definicja.wspolczynnikWzrostu * minuty * 0.5f;
        _wysokosc = Mathf.Clamp(_wysokosc + przyrost, 0f, definicja.maksymalnaWysokosc);
    }

    /// <summary>
    /// Aktualizuje skalę modelu 3D i efekty wizualne.
    /// </summary>
    void OdswiezModel()
    {
        float postep = _wysokosc / definicja.maksymalnaWysokosc;
        model3D.localScale = Vector3.one * postep;

        // Efekt kwitnienia po przekroczeniu progu
        bool powinienkwitnac = postep >= definicja.progKwitnienia;
        if (powinienkwitnac && !_kwitnie)
        {
            _kwitnie = true;
            efektKwitnienia?.Play();
        }
        else if (!powinienkwitnac && _kwitnie)
        {
            _kwitnie = false;
            efektKwitnienia?.Stop();
        }
    }

    /// <summary>
    /// Podlewa roślinę - wywoływane przez ShakeDetector.
    /// </summary>
    public void Podlej()
    {
        double minutyOdPodlania = TimeManager.MinutyOdZapisu(KluczPodlew);
        bool możnaPodleć = minutyOdPodlania >= definicja.interwaPodlewania * 60;

        if (!możnaPodleć)
        {
            Debug.Log("Roślina nie potrzebuje jeszcze wody.");
            return;
        }

        // Bonus wzrostu po podlaniu
        _wysokosc = Mathf.Clamp(
            _wysokosc + definicja.bonusWzrostuPoPodle,
            0f,
            definicja.maksymalnaWysokosc
        );

        TimeManager.ZapiszTeraz(KluczPodlew);
        OdswiezModel();
        Debug.Log($"{definicja.nazwaRosliny} podlana! Wysokość: {_wysokosc:F2} m");
    }
}

```

7. Sterowanie przez sensory IMU: akcelerometr i żyroskop

W grze AR Garden sensory IMU pełnią aktywną rolę w mechanikach: akcelerometr wykrywa potrząśnięcie (podlewanie), żyroskop kontroluje kąt padania słońca, a oba razem umożliwiają przechylenie wirtualnej podłogi, na której stoją rośliny. IMUSensorManager centralizuje odczyty i udostępnia przetworzone dane pozostałym skryptom.

6.1. IMUSensorManager

IMUSensorManager - centralne zarządzanie akcelerometrem i żyroskopem

```
using UnityEngine;
using System;

/// <summary>
/// Centralizuje odczyty akcelerometru i żyroskopu.
/// Udostępnia filtrowane wartości i zdarzenia gestów.
/// </summary>
public class IMUSensorManager : MonoBehaviour
{
    // Zdarzenia
    public static event Action OnPotrzezasniecie;
    public static event Action<float> OnZmianaPrzechylenia; // stopnie od -90 do 90

    // Właściwości odczytywane przez inne skrypty
    public static Vector3 FiltrowanyAkcel { get; private set; }
    public static Quaternion OrientacjaGyroskop { get; private set; }
    public static float PrzechylenieX { get; private set; }
    public static float PrzechylenieY { get; private set; }

    [Header("Ustawienia akcelerometru")]
    [SerializeField] private float wspolczynnikLP = 0.15f; // filtr dolnoprzepustowy
    [SerializeField] private float progPotrzezasniecia = 2.5f; // g
    [SerializeField] private float cooldownPotrzezas = 1.5f; // sekundy

    [Header("Ustawienia żyroskopu")]
    [SerializeField] private bool uzyjGyroskop = true;

    private bool _gyroObslugiwany;
    private float _ostatniePotrzezasniecie;
    private Vector3 _poprzedniAkcel;

    void Start()
    {
        _gyroObslugiwany = SystemInfo.supportsGyroscope;
        if (_gyroObslugiwany && uzyjGyroskop)
        {
            Input.gyro.enabled = true;
            Input.gyro.updateInterval = 0.0167f; // ~60 Hz
        }
    }

    void Update()
    {
        AktualizujAkcel();
        WykryjPotrzezasniecie();
        if (_gyroObslugiwany && uzyjGyroskop)
            AktualizujGyroskop();
    }

    void AktualizujAkcel()
    {
        Vector3 surowy = Input.acceleration;
```

```

        FiltrowanyAkcel = Vector3.Lerp(FiltrowanyAkcel, surowy, wspolczynnikLP);

        // Przechylenie w stopniach
        PrzechylenieX = Mathf.Asin(Mathf.Clamp(FiltrowanyAkcel.x, -1f, 1f)) * Mathf.Rad2Deg;
        PrzechylenieY = Mathf.Asin(Mathf.Clamp(FiltrowanyAkcel.y, -1f, 1f)) * Mathf.Rad2Deg;
    }

    void WykryjPotrzaskanie()
    {
        // Różnica między bieżącą a poprzednią klatką (surowy sygnał)
        Vector3 delta = Input.acceleration - _poprzedniAkcel;
        _poprzedniAkcel = Input.acceleration;

        bool wystarczajacaSila = delta.magnitude > progPotrzaskania;
        bool minelCooldown = Time.time - _ostatniePotrzaskanie > cooldownPotrzask;

        if (wystarczajacaSila && minelCooldown)
        {
            _ostatniePotrzaskanie = Time.time;
            OnPotrzaskanie?.Invoke();
            Debug.Log("Wykryto potrząśnięcie!");
        }
    }

    void AktualizujGyroskop()
    {
        Quaternion q = Input.gyro.attitude;
        // Konwersja układu współrzędnych czujnika do Unity
        OrientacjaGyroskop = new Quaternion(q.x, q.y, -q.z, -q.w);
    }
}

```

6.2. Podłączenie podlewania do IMU

Zdarzenie `OnPotrzaskanie` emitowane przez `IMUSensorManager` jest odbierane przez `ARGardenManager`, który podlewa najbliższą lub wszystkie rośliny w zasięgu.

ARGardenManager odbierający zdarzenie potrząśnięcia i podlewający rośliny

```

using UnityEngine;
using System.Collections.Generic;

/// <summary>
/// Odbiera zdarzenia z IMUSensorManager i przekazuje je roślinom.
/// </summary>
public class ARGardenManager : MonoBehaviour
{
    private readonly List<PlantController> _rosliny = new();

    void OnEnable()
    {
        IMUSensorManager.OnPotrzaskanie += PodlejWszystkie;
    }

    void OnDisable()
    {
        IMUSensorManager.OnPotrzaskanie -= PodlejWszystkie;
    }

    public void ZarejestrujRosline(PlantController roslina)
    {
        if (!_rosliny.Contains(roslina))
            _rosliny.Add(roslina);
    }

    public void WyrejestrujRosline(PlantController roslina)
    {
        _rosliny.Remove(roslina);
    }
}

```

```

void PodleJWszystkie()
{
    foreach (var roslina in _rosliny)
        roslina.Podlej();

    Debug.Log($"Podlano {_rosliny.Count} roślin przez potrząśnięcie.");
}
}

```

6.3. Kontrola kąta słońca żyroskopem

Gracz może zmieniać kąt padania wirtualnego światła przez przechylenie urządzenia. Gdy żyroskop nie jest dostępny, fallback korzysta z przechylenia akcelerometrem.

```

Sterowanie kątem słońca przez przechylenie urządzenia z płynnym wygładzaniem
using UnityEngine;

/// <summary>
/// Steruje kątem padania słońca przez orientację urządzenia.
/// Tryb żyroskopu: pełna orientacja. Tryb akcelerometru: tylko oś X.
/// </summary>
public class KontrolerSlonca : MonoBehaviour
{
    [SerializeField] private Light      swiatloKierunkowe;
    [SerializeField] private float      wplywGracza      = 0.3f; // 0..1
    [SerializeField] private float      predkoscObrotu    = 45f; // stopnie/s
    [SerializeField] private bool       uzyjGyroskop      = true;

    // Bazowy kat wynikajacy z pory dnia (z SlonceAR)
    private float _katBazowy;
    // Offset sterowany przez gracza
    private float _katOffset;

    void Update()
    {
        // Aktualizacja kata bazowego z pory dnia
        float tDoby = TimeManager.TDoby;
        _katBazowy = Mathf.Lerp(-90f, 270f, tDoby);

        // Odczyt sterowania gracza
        float sterowanie = 0f;

        if (uzyjGyroskop && SystemInfo.supportsGyroscope)
        {
            // Przechylenie w osi X z żyroskopu (uchwyt poziomy)
            float przechylenieGracz = IMUSensorManager.PrzechylenieX;
            sterowanie = przechylenieGracz * wplywGracza;
        }
        else
        {
            // Fallback: akcelerometr
            sterowanie = IMUSensorManager.FiltrowanyAkcel.x * 30f * wplywGracza;
        }

        // Wygładzone przejście do docelowego offsetu
        float docelowy = Mathf.Clamp(sterowanie, -60f, 60f);
        _katOffset = Mathf.MoveTowards(_katOffset, docelowy,
                                       predkoscObrotu * Time.deltaTime);

        // Zastosowanie łącznego kata
        float katKoncowy = _katBazowy + _katOffset;
        swiatloKierunkowe.transform.rotation = Quaternion.Euler(katKoncowy, -30f, 0f);
    }
}

```

8. Dynamiczne obiekty AR reagujące na ruch urządzenia

Oprócz sterowania światłem i podlewania, sensory IMU wpływają na zachowanie obiektów w przestrzeni AR. Delikatne przechylenie urządzenia powoduje kołysanie gałęzi roślin, symulując wiatr. Gwałtowny ruch aktywuje efekty cząsteczkowe podlewania.

7.1. Symulacja wiatru przez IMU

Efekt kołysania gałęzi przez przechylenie urządzenia

```
using UnityEngine;

/// <summary>
/// Kołysa obiektem (np. gałęzią rośliny) na podstawie
/// przechylenia urządzenia. Tworzy wrażenie wiatru.
/// </summary>
public class EfektWiatru : MonoBehaviour
{
    [SerializeField] private float amplituda = 5f; // maksymalny kąt (stopnie)
    [SerializeField] private float wygladz = 4f; // szybkość wygładzania

    private Quaternion _rotacjaBazowa;
    private Vector3 _celowyOffset;

    void Start()
    {
        _rotacjaBazowa = transform.localRotation;
    }

    void Update()
    {
        // Przechylenie urządzenia jako wektor wiatru
        float wiatrX = -IMUSensorManager.PrzechylenieY * amplituda;
        float wiatrZ = IMUSensorManager.PrzechylenieX * amplituda;

        _celowyOffset = new Vector3(wiatrX, 0f, wiatrZ);

        // Wygładzone przejście
        Quaternion celowaRotacja = _rotacjaBazowa *
            Quaternion.Euler(_celowyOffset);

        transform.localRotation = Quaternion.Slerp(
            transform.localRotation,
            celowaRotacja,
            Time.deltaTime * wygladz
        );
    }
}
```

7.2. Efekt podlewania przez potrząśnięcie

Efekt podlewania wyzwalany zdarzeniem potrząśnięcia z systemu IMU

```
using UnityEngine;

/// <summary>
/// Wyświetla efekt cząsteczkowy wody i animację konewki
/// w momencie wykrycia potrząśnięcia urządzenia.
/// </summary>
public class EfektPodlewania : MonoBehaviour
{
    [SerializeField] private ParticleSystem cząsteczkiWody;
    [SerializeField] private Animator animatorKonewki;
    [SerializeField] private AudioSource dzwiekWody;
```

```

[SerializeField] private float      czasEfektu = 2f;

void OnEnable()
{
    IMUSensorManager.OnPotrzebnosc += AktywujEfekt;
}

void OnDisable()
{
    IMUSensorManager.OnPotrzebnosc -= AktywujEfekt;
}

void AktywujEfekt()
{
    cząsteczkiWody?.Play();
    animatorKonevki?.SetTrigger("Podlej");
    dzwiekWody?.PlayOneShot(dzwiekWody.clip);

    // Zatrzymanie efektu po czasie
    CancelInvoke(nameof(ZatrzymajEfekt));
    Invoke(nameof(ZatrzymajEfekt), czasEfektu);
}

void ZatrzymajEfekt()
{
    cząsteczkiWody?.Stop();
}
}

```

7.3. Umieszczanie roślin w przestrzeni AR

Gracz sadi rośliny przez dotknięcie ekranu, analogicznie do wzorca tap-to-place z Laboratorium Unity 5. Każda roślina jest zakotwiczona do wykrytej płaszczyzny i rejestrowana w ARGardenManager.

SadzarkaRoslin - umieszczanie roślin z zakotwiczaniem w przestrzeni AR

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using UnityEngine.InputSystem;

/// <summary>
/// Sadi rośliny w scenie AR przez dotyknięcie ekranu.
/// Każda roślina jest instancją prefabrykatu z PlantController.
/// </summary>
[RequireComponent(typeof(ARRaycastManager))]
[RequireComponent(typeof(ARAnchorManager))]
public class SadzarkaRoslin : MonoBehaviour
{
    [SerializeField] private ARGardenManager ogrod;
    [SerializeField] private ARPlaneManager planeManager;
    [SerializeField] private PlantSO      wybranaRoslina;
    [SerializeField] private int          maxRoslin = 6;

    private ARRaycastManager _raycastManager;
    private ARAnchorManager _anchorManager;
    private List<ARRaycastHit> _trafienia = new();
    private int _liczbaRoslin;

    void Awake()
    {
        _raycastManager = GetComponent<ARRaycastManager>();
        _anchorManager = GetComponent<ARAnchorManager>();
    }

    void Update()
    {

```

```

        if (Touchscreen.current == null) return;
        var dotyk = Touchscreen.current.primaryTouch;
        if (!dotyk.press.wasPressedThisFrame) return;
        if (wybranaRoslina == null) return;
        if (_liczbaRoslin >= maxRoslin) return;

        Vector2 poz = dotyk.position.ReadValue();

        bool trafiono = _raycastManager.Raycast(
            poz, _trafienia, TrackableType.PlaneWithinPolygon);
        if (!trafiono) return;

        Pose pozaAR = _trafienia[0].pose;

        // Zakotwicz roślinę
        ARAnchor kotwa = _anchorManager.AttachAnchor(
            (ARPlane)_trafienia[0].trackable, pozaAR);
        if (kotwa == null) return;

        // Instancja prefabrykatu rośliny jako dziecko kotwicy
        var instancja = Instantiate(wybranaRoslina.prefabAR,
            kotwa.transform.position,
            Quaternion.identity);
        instancja.transform.SetParent(kotwa.transform, true);

        // Rejestracja w menedżerze ogrodu
        var kontroler = instancja.GetComponent<PlantController>();
        if (kontroler != null)
            ogrod.ZarejestujRosline(kontroler);

        _liczbaRoslin++;
        Debug.Log($"Posadzono: {wybranaRoslina.nazwaRosliny}
        ({_liczbaRoslin}/{maxRoslin})");
    }

    public void UstawRosline(PlantSO roslina) => wybranaRoslina = roslina;
}

```

9. System nagród oparty na kalendarzu i zdarzeniach specjalnych

System nagród wzmacnia zaangażowanie gracza przez powiązanie atrakcji w grze z rzeczywistym kalendarzem. Święta, pory roku i codzienne logowanie nagradzane są specjalnymi roślinami, ziarnami i efektami wizualnymi.

8.1. Obsługa zdarzeń kalendarzowych w UI

Panel UI zdarzenia kalendarzowego z odbieraniem nagrody

```

using UnityEngine;
using TMPro;
using UnityEngine.UI;

/// <summary>
/// Wyświetla panel powiadomienia o aktywnym zdarzeniu kalendarzowym.
/// Subskrybuje zdarzenie z CalendarManager.
/// </summary>
public class UIZdarzeniaKalendar : MonoBehaviour
{
    [SerializeField] private GameObject panelZdarzenia;
    [SerializeField] private TextMeshProUGUI txtTytuł;
    [SerializeField] private TextMeshProUGUI txtOpis;
    [SerializeField] private Image imgBaner;
    [SerializeField] private TextMeshProUGUI txtNagroda;
}

```

```

[SerializeField] private Button      btnOdbierz;

private CalendarEventSO _aktualneZdarzenie;

void OnEnable()
{
    CalendarManager.NaZdarzenieKalendarzowe += PokazZdarzenie;
}

void OnDisable()
{
    CalendarManager.NaZdarzenieKalendarzowe -= PokazZdarzenie;
}

void PokazZdarzenie(CalendarEventSO zdarzenie)
{
    _aktualneZdarzenie = zdarzenie;

    txtTytul.text = zdarzenie.tytulZdarzenia;
    txtOpis.text  = zdarzenie.opisZdarzenia;

    if (zdarzenie.baner != null)
        imgBaner.sprite = zdarzenie.baner;

    string opisNagrody = zdarzenie.bonusZiaren > 0
        ? $"#{zdarzenie.bonusZiaren} ziaren"
        : string.Empty;

    if (zdarzenie.odblokowananaroslina != null)
        opisNagrody += $" + {zdarzenie.odblokowananaroslina.nazwaRosliny}!";

    txtNagroda.text = opisNagrody;
    btnOdbierz.onClick.RemoveAllListeners();
    btnOdbierz.onClick.AddListener(OdbierzNagrode);

    panelZdarzenia.SetActive(true);
}

void OdbierzNagrode()
{
    if (_aktualneZdarzenie == null) return;

    GameManager.Instance.DodajZiarna(_aktualneZdarzenie.bonusZiaren);

    // Odblokowanie specjalnej rośliny
    if (_aktualneZdarzenie.odblokowananaroslina != null)
        Debug.Log($"Odblokowano:
{_aktualneZdarzenie.odblokowananaroslina.nazwaRosliny}");

    panelZdarzenia.SetActive(false);
    _aktualneZdarzenie = null;
}
}

```

8.2. Nagroda za serię dziennych logowań

Im dłuższa seria logowań, tym większa nagroda dzienna. Mnożnik nagród rośnie nieliniowo, motywując gracza do codziennego powrotu do gry:

Kalkulator nagród za serię dziennych logowań

```

/// <summary>
/// Oblicza nagrodę za serię dziennych logowań.
/// Bazowa nagroda rośnie logarytmicznie z serią.
/// </summary>
public static class KalkulatorNagrody
{
    private const int BAZA_ZIARNA = 10;

```

```

public static int ObliczZiarnaSerii(int seria)
{
    // Wzor: baza * log2(seria + 1) zaokrąglony do gore
    float mnoznik = Mathf.Log(seria + 1, 2f) + 1f;
    return Mathf.CeilToInt(BAZA_ZIARNA * mnoznik);
}

public static string OpisNagrody(int seria)
{
    int ziarna = ObliczZiarnaSerii(seria);

    return seria switch
    {
        1      => $"Dzień 1! Nagroda startowa: {ziarna} ziaren.",
        7      => $"Tydzień! Bonus: {ziarna} ziaren + specjalna roślina!",
        30     => $"Miesiąc! Nagroda mistrzowska: {ziarna} ziaren!",
        -      => $"Dzień {seria} z rzędu! Nagroda: {ziarna} ziaren."
    };
}
}

```

8.3. Predefiniowane zdarzenia kalendarzowe

Tabela zawiera przykładowe zdarzenia do skonfigurowania jako pliki CalendarEventSO w projekcie.

Dzień	Miesiąc	Nazwa zdarzenia	Nagroda
1	1	Nowy Rok	Roślina ogniowa + 50 ziaren
14	2	Walentynki	Róża AR + 30 ziaren
1	3	Początek wiosny	Tulipan + 40 ziaren
1	6	Dzień Dziecka	Balonowe kwiaty + 25 ziaren
21	6	Letnie przesilenie	Słonecznik AR + 35 ziaren
1	11	Wszystkich świętych	Drzewo jesieni + 20 ziaren
6	12	Mikołajki	Śnieżna choinka + 45 ziaren
25	12	Boże Narodzenie	Zimowy ogród + 100 ziaren
0	0	Urodziny gracza	Tort AR + 200 ziaren (wpisuje datę sam)

WSKAZOWKA

Zdarzenie z dzień = 0 i miesiąc = 0 jest interpretowane jako "każdy dzień każdego miesiąca" w metodzie CzyAktywne(). Urodziny gracza wymagają osobnej obsługi: pobrania daty urodzin z PlayerPrefs i porównania z DateTime.Now.Day i DateTime.Now.Month.

10. Interfejs użytkownika i HUD gry

HUD (Heads-Up Display) w grze AR musi być czytelny na tle rzeczywistego świata oraz nie zasłaniać kluczowych elementów sceny. Stosuje się przezroczyste panele w narożnikach ekranu, ikony zamiast długich napisów oraz animowane wskaźniki stanu.

9.1. Komponenty HUD

Element HUD	Położenie	Wyświetlana informacja
-------------	-----------	------------------------

Pasek zegara	Góra środek	Godzina rzeczywista + pora dnia (ikona)
Licznik ziaren	Góra prawo	Aktualna liczba ziaren + ikona
Pasek serii	Góra lewo	Liczba dni z rzędu (płomień)
Pasek podlewania	Dół środek	Cooldown podlewania (okrągły progress)
Panel wyboru rośliny	Dół	Dostępne gatunki do posadzenia
Przycisk ustawienia	Góra prawo	Otwarcie menu pauzy

9.2. Kontroler HUD

HUDKontroler - zegar rzeczywisty, zasoby, ikona pory dnia i cooldown podlewania

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using System;

/// <summary>
/// Aktualizuje elementy HUD na podstawie stanu gry i czasu systemowego.
/// </summary>
public class HUDKontroler : MonoBehaviour
{
    [Header("Wyświetlacze czasu")]
    [SerializeField] private TextMeshProUGUI txtGodzina;
    [SerializeField] private Image imgIkonaPory;
    [SerializeField] private Sprite[] spritesPorDnia; // 4: Swit, Dzień, Zmierzch,
    Noc

    [Header("Zasoby gracza")]
    [SerializeField] private TextMeshProUGUI txtZiarna;
    [SerializeField] private TextMeshProUGUI txtSeria;

    [Header("Podlewanie")]
    [SerializeField] private Image kregCooldown; // fillAmount 0..1
    [SerializeField] private TextMeshProUGUI txtPodlewanie;

    [Header("Konfiguracja")]
    [SerializeField] private PlantSO roslina; // aktualnie wybrana
    [SerializeField] private float maxCooldown = 3600f; // sekundy

    private float _cooldownPodlew;

    void OnEnable()
    {
        TimeManager.NaZmianiePoryDnia += AktualizujIkonePory;
        IMUSensorManager.OnPotrzebie += ResetujCooldown;
    }

    void OnDisable()
    {
        TimeManager.NaZmianiePoryDnia -= AktualizujIkonePory;
        IMUSensorManager.OnPotrzebie -= ResetujCooldown;
    }

    void Update()
    {
        // Zegar rzeczywisty
        txtGodzina.text = DateTime.Now.ToString("HH:mm");

        // Zasoby
        txtZiarna.text = GameManager.Instance.Ziarna.ToString("N0");
    }
}
```

```

        txtSeria.text = $"{GameManager.Instance.SeriaDni} dni";

        // Cooldown podlewania
        _cooldownPodlew = Mathf.Max(0f, _cooldownPodlew - Time.deltaTime);
        float postep = 1f - _cooldownPodlew / maxCooldown;
        kregCooldown.fillAmount = postep;

        bool gotowy = _cooldownPodlew <= 0f;
        txtPodlewanie.text = gotowy
            ? "Potrząśnij, aby podlać!"
            : TimeSpan.FromSeconds(_cooldownPodlew).ToString(@"mm\:ss");
    }

    void AktualizujIkonePory(PoraDnia pora)
    {
        int idx = (int)pora;
        if (idx >= 0 && idx < spritesPorDnia.Length)
            imgIkonaPory.sprite = spritesPorDnia[idx];
    }

    void ResetujCooldown()
    {
        _cooldownPodlew = maxCooldown;
    }
}

```

11. Budowanie i testowanie gry na urządzeniu mobilnym

Kompletna gra AR Garden wymaga fizycznego urządzenia do pełnego przetestowania. W edytorze możliwe jest testowanie logiki czasu, kalendarza i IMU (przez Device Simulator), lecz AR Foundation działa pełnoprawnie tylko na fizycznym telefonie.

10.1. Konfiguracja Build Settings

Przed zbudowaniem projektu należy upewnić się, że wszystkie trzy sceny są zarejestrowane w Build Settings w kolejności: BootScene (0), MenuScene (1), GameScene (2). Platforma docelowa to Android lub iOS.

Ustawienie Player Settings	Wartość dla AR Garden
Product Name	AR Garden
Package Name	com.uczelnia.argarden
Version	1.0.0
Minimum API Level	Android 9.0 (API 28)
Target API Level	Automatyczny lub Android 14 (API 34)
Scripting Backend	IL2CPP
Target Architectures	ARM64
Camera Usage Description	AR Garden używa kamery do wyświetlania ogrodu AR
Multithreaded Rendering	Włączone (lepsze FPS na ARM)
Graphics API	Vulkan (Android), Metal (iOS)

10.2. Symulacja czasu w edytorze

Testowanie mechanik opartych na czasie bez czekania na rzeczywistą zmianę godziny realizujemy przez tymczasowy skrypt edytorski przesłaniający `DateTime.Now`. Zalecane podejście to interfejs dostarczający czas.

Wzorec wstrzykiwania zależności (DI) dla źródła czasu - testowalny TimeManager

```
/// <summary>
/// Interfejs źródła czasu. Pozwala podmienić zegar na testowy.
/// </summary>
public interface IZrodloCzasu
{
    System.DateTime Teraz { get; }
}

/// <summary>
/// Produkcyjne źródło czasu: odczytuje DateTime.Now.
/// </summary>
public class SystemoweZrodloCzasu : IZrodloCzasu
{
    public System.DateTime Teraz => System.DateTime.Now;
}

/// <summary>
/// Testowe źródło czasu: stała wartość do testowania konkretnej pory dnia.
/// Umożliwia symulację np. pory nocnej w edytorze bez czekania.
/// </summary>
public class TestoweZrodloCzasu : IZrodloCzasu
{
    public System.DateTime Teraz { get; set; } = System.DateTime.Now;

    public TestoweZrodloCzasu UstawGodzine(int godzina, int minuta = 0)
    {
        var dziś = System.DateTime.Today;
        Teraz = new System.DateTime(dziś.Year, dziś.Month, dziś.Day,
                                    godzina, minuta, 0);
        return this;
    }
}

// Przykład użycia w scenie deweloperskiej:
// var zrodlo = new TestoweZrodloCzasu().UstawGodzine(22); // godzina 22:00
// TimeManager.InjektujZrodloCzasu(zrodlo);
```

10.3. Lista kontrolna przed buildem

Przed każdym zbudowaniem wersji testowej należy sprawdzić:

- Package Manager: AR Foundation, ARCore XR Plugin i ARKit XR Plugin są zainstalowane w zgodnych wersjach.
- XR Plug-in Management: ARCore (Android) i ARKit (iOS) są zaznaczone.
- Player Settings: Camera Usage Description wypełniona po polsku lub angielsku.
- Build Settings: trzy sceny wpisane w kolejności BootScene (0), MenuScene (1), GameScene (2).
- ScriptableObjects: wszystkie rośliny i zdarzenia kalendarzowe mają wypełnione pola.
- Keystore: plik podpisu istnieje i jest przypisany w Publishing Settings.
- Minimum API Level: ustawiony na Android 28 lub wyższy.
- Target Architectures: zaznaczony ARM64.

10.4. Testy funkcjonalne na urządzeniu

Test	Sposób weryfikacji
------	--------------------

Wykrywanie płaszczyzn	Podłoga i stół zostają wykryte w 15 s
Sadzenie roślin	Dotyk umieszcza roślinę na płaszczyźnie
Podlewanie	Potrząśnięcie wyzwala efekt wody
Wiatr	Przechylenie kołysze gałęziami roślin
Kąt słońca	Obrót zmienia kierunek cienia w scenie
Pora dnia	Ikona HUD i oświetlenie zgodne z godziną
Seria dzienna	Kolejne uruchomienie zwiększa licznik
Zdarzenie kalendarzowe	Dzisiaj aktywne zdarzenie wyświetla panel
Wzrost offline	Po 30 min pauzy roślina urasta o spodziewany przyrost
Safe Area	HUD nie zakrywa wcięcia kamery (notch)

12. Zadania dla studenta

Poniższe zadania należy wykonać samodzielnie jako rozbudowanie projektu AR Garden. Każde zadanie powinno być przetestowane w edytorze (XR Simulation) i opcjonalnie na fizycznym urządzeniu.

1. Konfiguracja projektu: załóż nowy projekt Unity 6 z Universal Render Pipeline i AR Foundation 6.x. Stworzenie hierarchii ScriptableObjects: minimum trzy rośliny (np. tulipan, słonecznik, kaktus) z różnymi preferowanymi porami roku, współczynnikami wzrostu i progami kwitnienia. Każda roślina powinna różnić się wizualnie prefabrykatem 3D.
2. TimeManager i pory dnia: zaimplementuj klasę TimeManager według wzoru z sekcji 4.1. Przetestuj poprawność przejść między porami dnia przez skrypt TestoweZrodloCzasu ustawiając kolejno godziny 6, 10, 18 i 23. Zweryfikuj, że zdarzenie NaZmianiePoryDnia wywoływane jest dokładnie raz przy każdej zmianie.
3. PlantController z wzrostem offline: zaimplementuj logikę wzrostu rośliny według sekcji 5.1. Roślina musi: rosnąć z mnożnikiem zależnym od pory dnia, rosnąć szybciej w ulubionej porze roku, doliczać wzrost za czas spędzony poza grą. Przetestuj przez symulację: ustaw testowy czas na noc i potwierdź, że roślina rośnie znacznie wolniej niż w południe.
4. IMU podlewanie: zaimplementuj IMUSensorManager z wykrywaniem potrząśnięcia (sekcja 6.1) oraz połącz go z funkcją Podle() w PlantController. Skonfiguruj próg potrząśnięcia i cooldown w Inspektorze. Dodaj efekt cząsteczkowy i dźwięk wody wyzwany przez IMUSensorManager.OnPotrzezasniecie.
5. CalendarManager i zdarzenia: skonfiguruj co najmniej pięć plików CalendarEventSO (w tym jedno zdarzenie na dzień dzisiejszy do testów). Zaimplementuj CalendarManager sprawdzający aktywne zdarzenia i UIZdarzeniaKalendar wyświetlający panel z nagrodą. Sprawdź, że zdarzenie pojawia się automatycznie po uruchomieniu aplikacji w dniu zdarzenia.
6. Kompletna scena AR: połącz wszystkie elementy w działającą scenę AR: wykrywanie płaszczyzn, sadzenie roślin przez dotyk (tap-to-place z zakotwiczeniem), wzrost zależny od czasu, podlewanie przez potrząśnięcie, HUD z zegarem i licznikiem ziaren. Zbuduj plik APK i przetestuj wszystkie mechaniki na fizycznym urządzeniu Android. Zmierz FPS przez Unity Profiler i upewnij się, że jest wyższy niż 30.

13. Zadania dodatkowe

Poniższe zadania mają charakter badawczy i projektowy. Wymagają samodzielnego eksplorowania dokumentacji Unity, ARCore i ARKit oraz kreatywnego zastosowania poznanych technik.

- **Dynamiczna pogoda AR:** zaimplementuj system pogodowy oparty na miesiącu roku. Wiosna: efekt deszczu (cząsteczki) z góry sceny AR. Lato: fala gorąca (shader zniekształcający). Jesień: opadające liście (cząsteczki). Zima: śnieg i białe tło mrozu na krawędziach ekranu. Efekty cząsteczkowe powinny być zakotwiczone ponad sceną AR i reagować na zmianę miesiąca.
- **Urodziny gracza:** rozbuduj CalendarManager o obsługę daty urodzin gracza. Przy pierwszym uruchomieniu poproś gracza o podanie daty urodzin (DatePicker lub trzy InputField). Zapisz datę w PlayerPrefs. W dniu urodzin wyświetl wyjątkowy panel AR z torciem, fajerwerkami i nagrodą 200 ziaren. Tort powinien być obiektem 3D zakotwiczonym na płaszczyźnie.
- **Sklep roślin z inną walutą:** zaimplementuj pełen ekran sklepu w MenuScene. Lista dostępnych roślin pochodzi z tablicy PlantSO. Cena w ziarnach odczytywana jest z pola cenaZiaren. Kupione rośliny zapisywane są w PlayerPrefs jako JSON (lista nazw). W scenie AR dostępne są tylko rośliny zakupione w sklepie lub odblokowane przez zdarzenia kalendarzowe.
- **Wieloosobowy ogród AR (Cloud Anchors):** dwóch graczy widzi te same rośliny w tym samym miejscu w przestrzeni rzeczywistej. Użyj ARCore Cloud Anchors do zsynchronizowania zakotwic. Gracz A sadi roślinę i udostępnia kod zakotwicy. Gracz B wpisuje kod i widzi tę samą roślinę we własnej sesji AR.
- **Powiadomienia o podlewaniu:** skonfiguruj lokalne powiadomienia (Unity Mobile Notifications - Lab 4) wysyłane automatycznie, gdy rośliny wymagają podlania. System powinien harmonogramować powiadomienie przy zamknięciu aplikacji z czasem odkładanym równym interwałem podlewania rośliny. Treść powiadomienia powinna zawierać nazwę rośliny wymagającej uwagi.

-

Przydatne zasoby do zadań dodatkowych:

- Dokumentacja: [AR Foundation - dokumentacja](#)
- Dokumentacja: [ARCore Cloud Anchors](#)
- Dokumentacja: [Unity Mobile Notifications](#)
- Dokumentacja: [Netcode for GameObjects](#)
- Dokumentacja: [Unity Particle System](#)
- Dokumentacja: [System.DateTime - dokumentacja MSDN](#)