POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

KATEDRA INFORMATYKI I AUTOMATYKI
POLITECHNIKI RZESZOWSKIEJ

Instrukcja laboratoryjna

Programowanie aplikacji mobilnych

Laboratorium 1

UI/UX Design, wygląd aplikacji oraz nawigacja pomiędzy oknami na przykładzie projektu 2D Mobile w środowisku Unity

dr inż. Mateusz Pomianek

05.03.2025

Spis treści

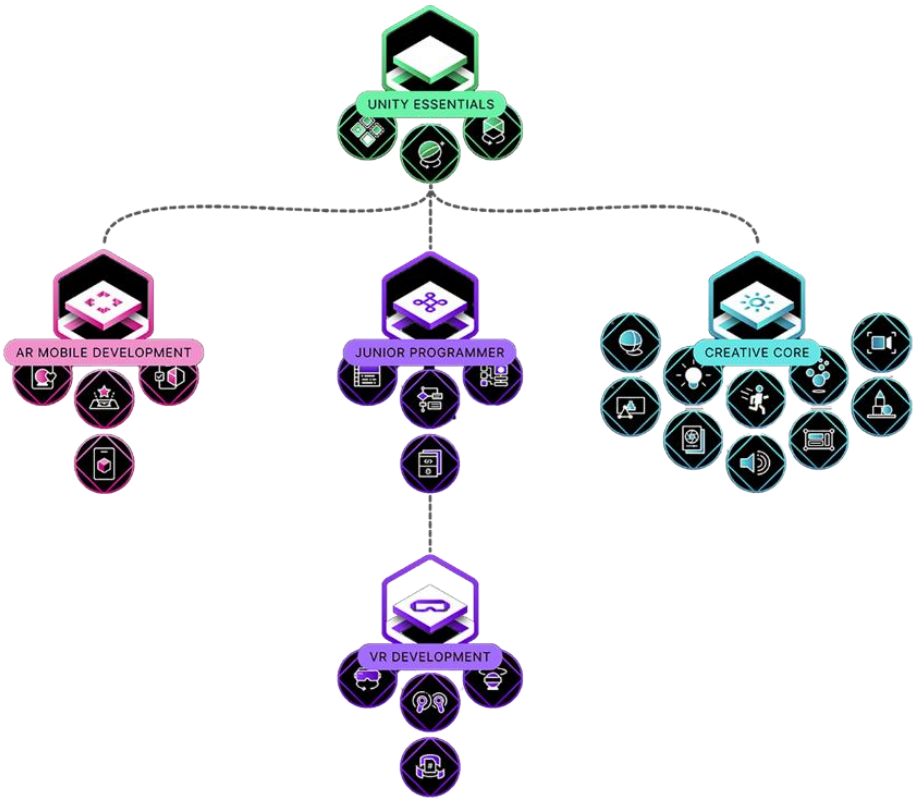
1. Środowisko edytora unity

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

Rys. 1. Ikona 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.

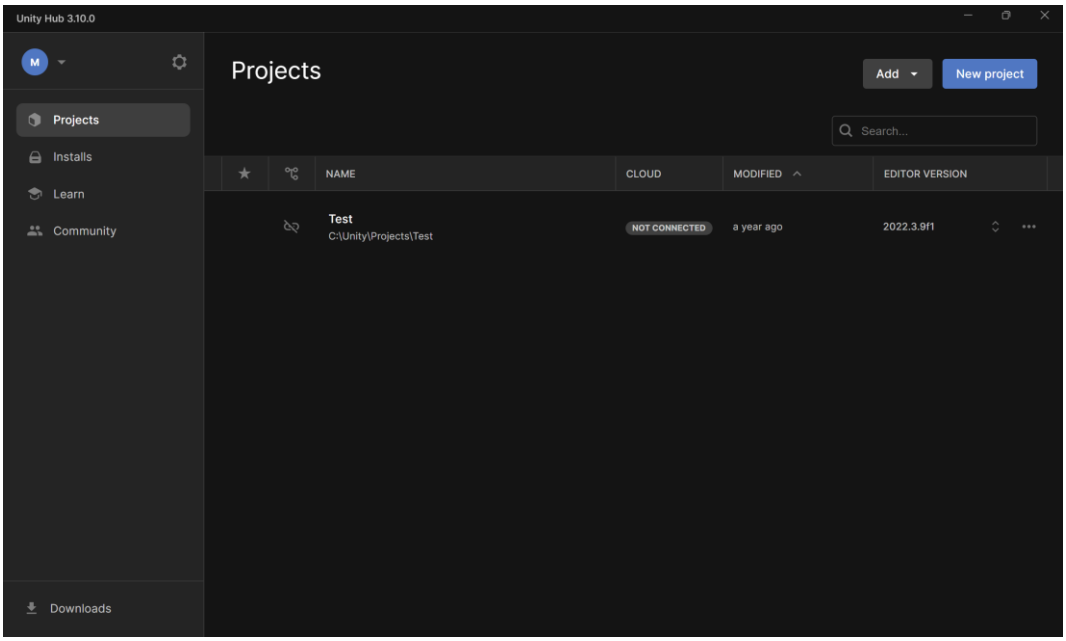


Rys. 2. Wykres poszczególnych aplikacji edytora Unity

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.

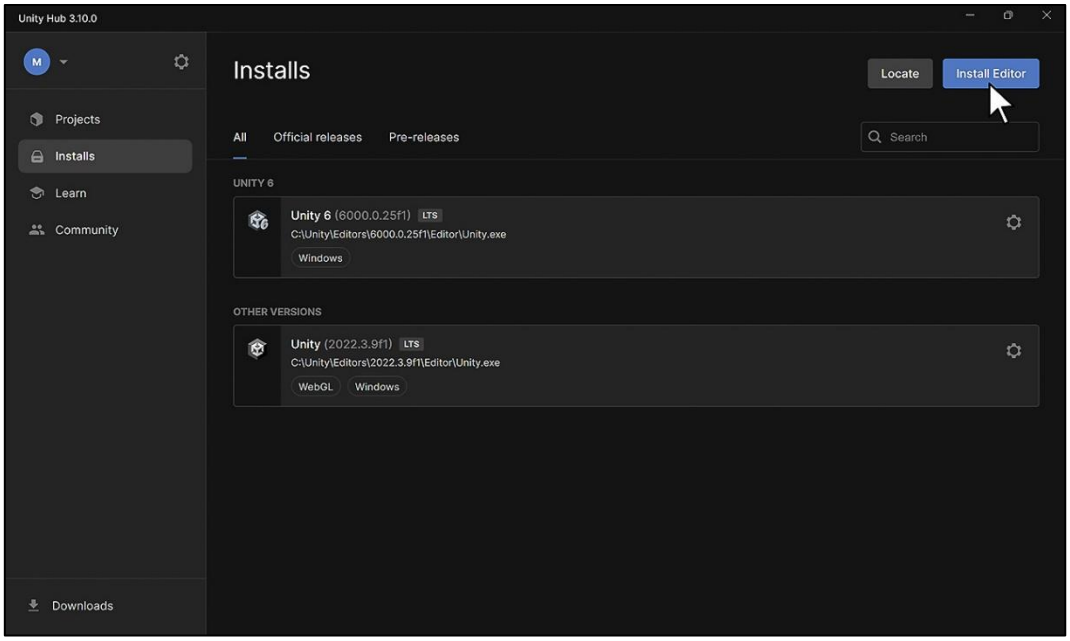
2. Tworzenie nowego projektu

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.



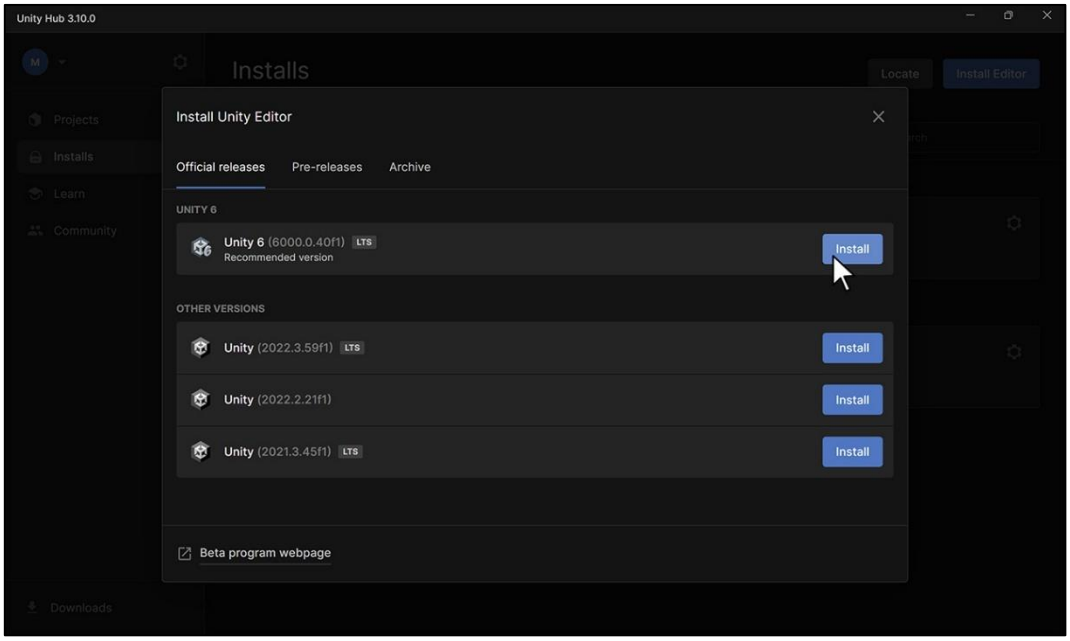
Rys. 3. Widok projektów edytora Unity

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



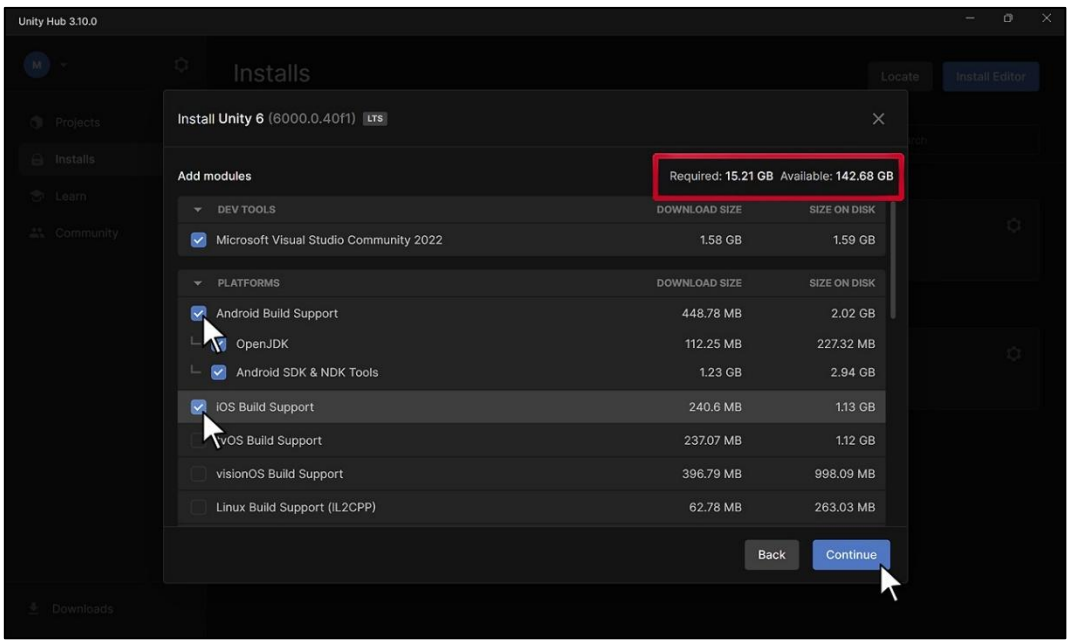
Rys. 4. Zainstalowane wersje edytora Unity

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



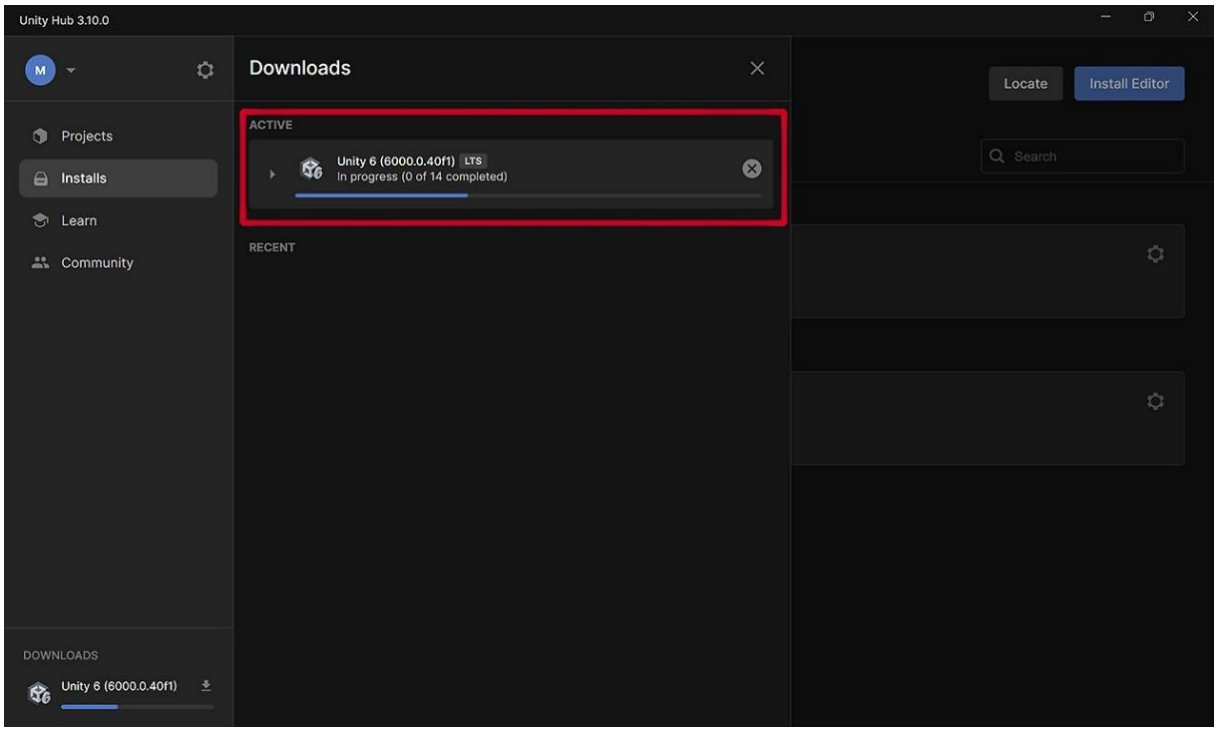
Rys. 5. Wybór wersji edytora do zainstalowania

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.



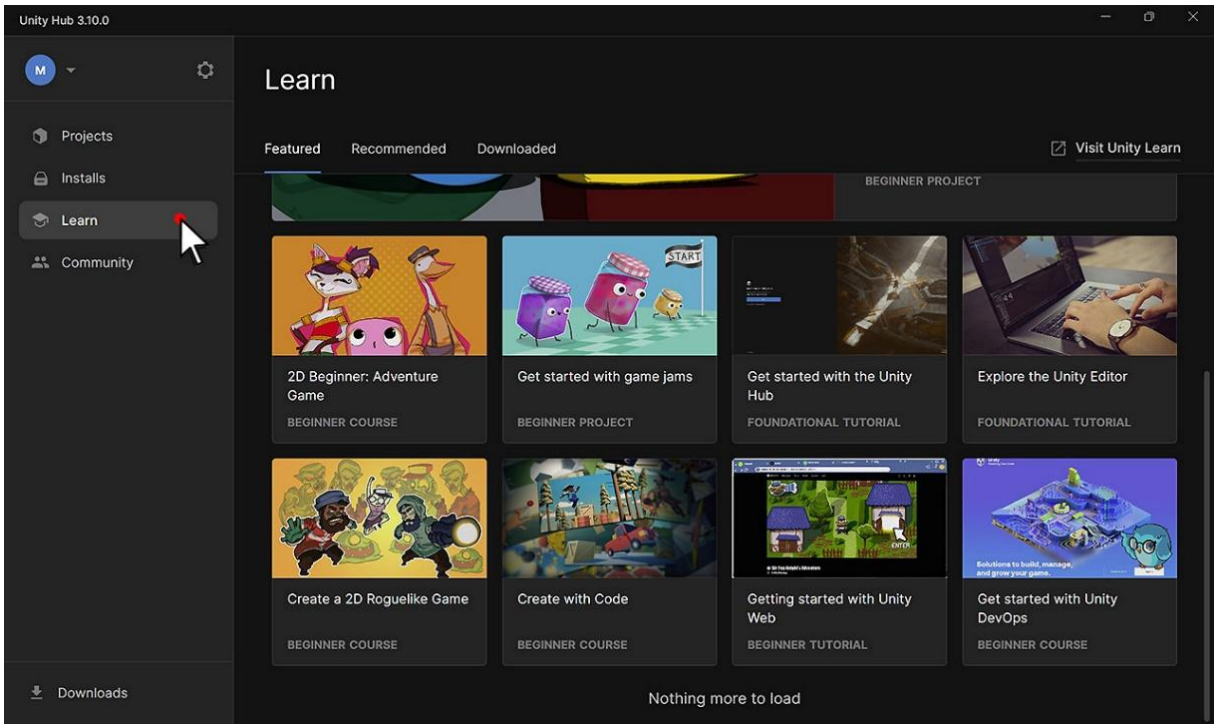
Rys. 6. Wybór dodatkowych pakietów instalowanych wraz z nową wersją edytora Unity

Następnie akceptujemy wszystkie kolejne zgody i oczekujemy na zakończenie instalacji.

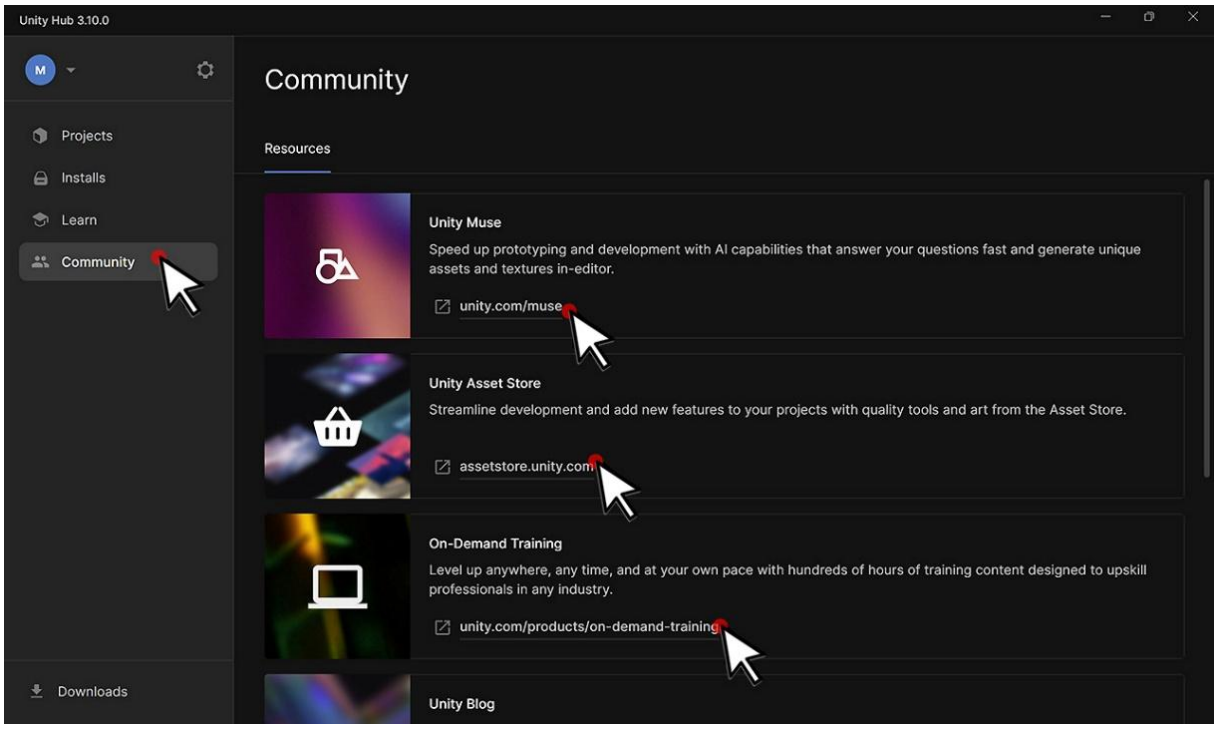


Rys. 7. Okno instalacji edytora Unity

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



Rys. 8. Zakładka Learn



Rys. 9. Zakładka Comunity

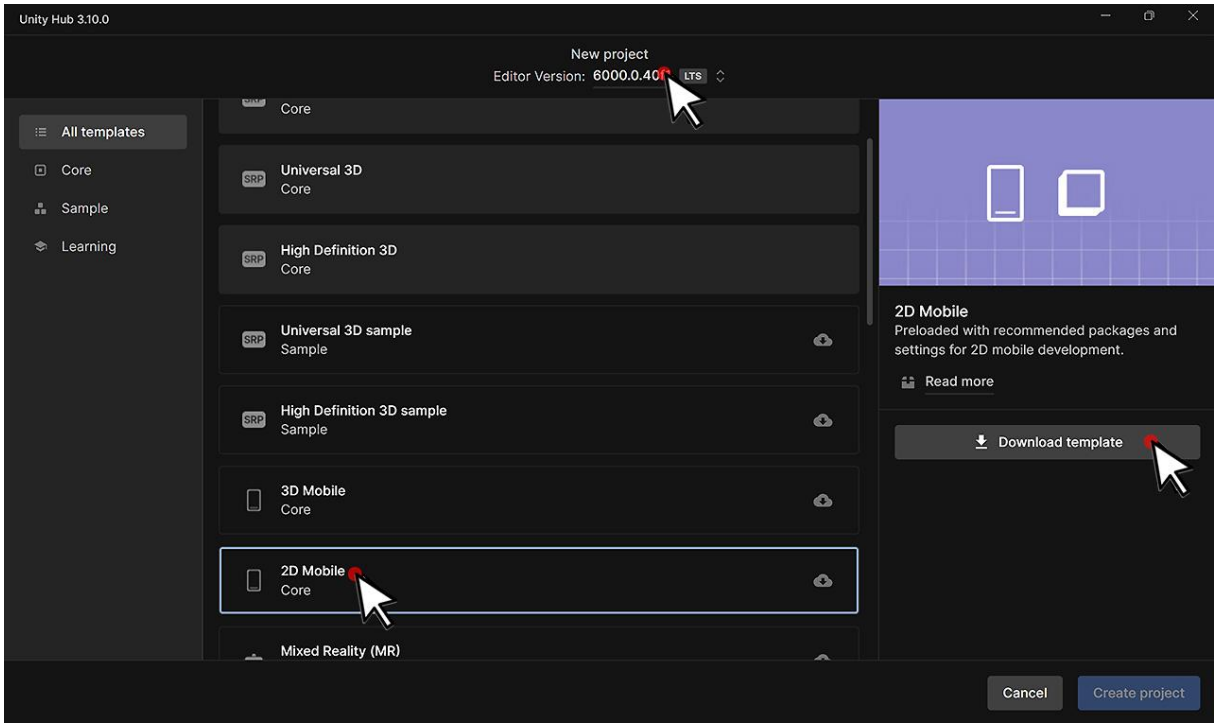
W zakładce **Learn** znajdują się ciekawe kursy i przykładowe projekty poruszające szerszy zakres budowania aplikacji niż opisany w tej instrukcji. Przykładowymi szkoleniami, mogą być:

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.

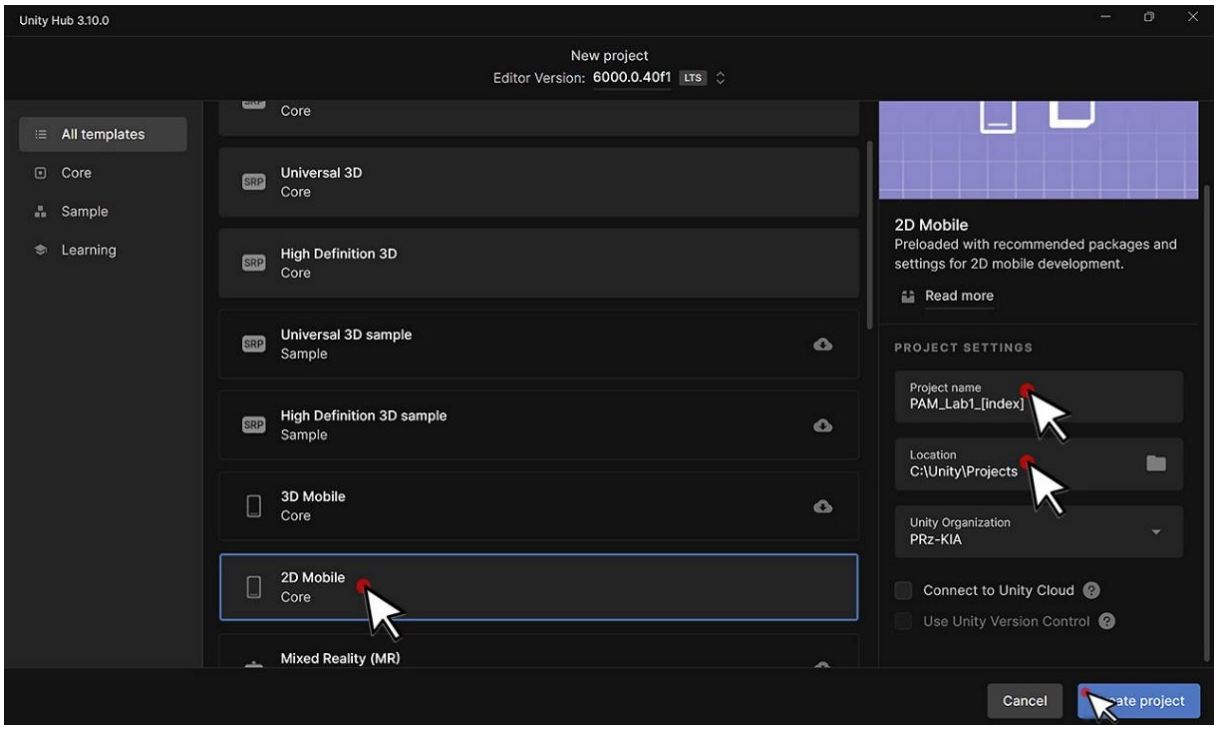
Gdy zainstalujemy interesującą nas wersję edytora Unity naciskamy przycisk **nowy projekt**, a następnie wybieramy **2D Mobile** oraz pobieramy **Template** dotyczący tego typu projektu.

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.

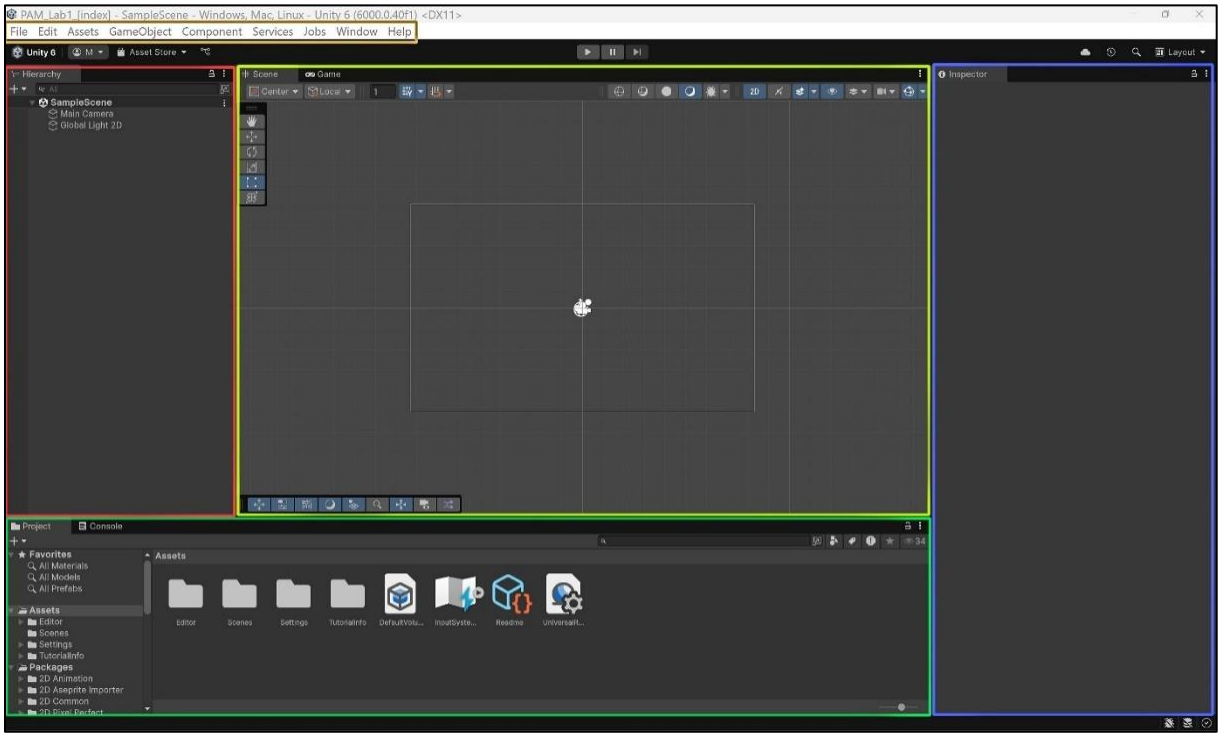


Rys. 11. Wybór nazwy i lokalizacji

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:

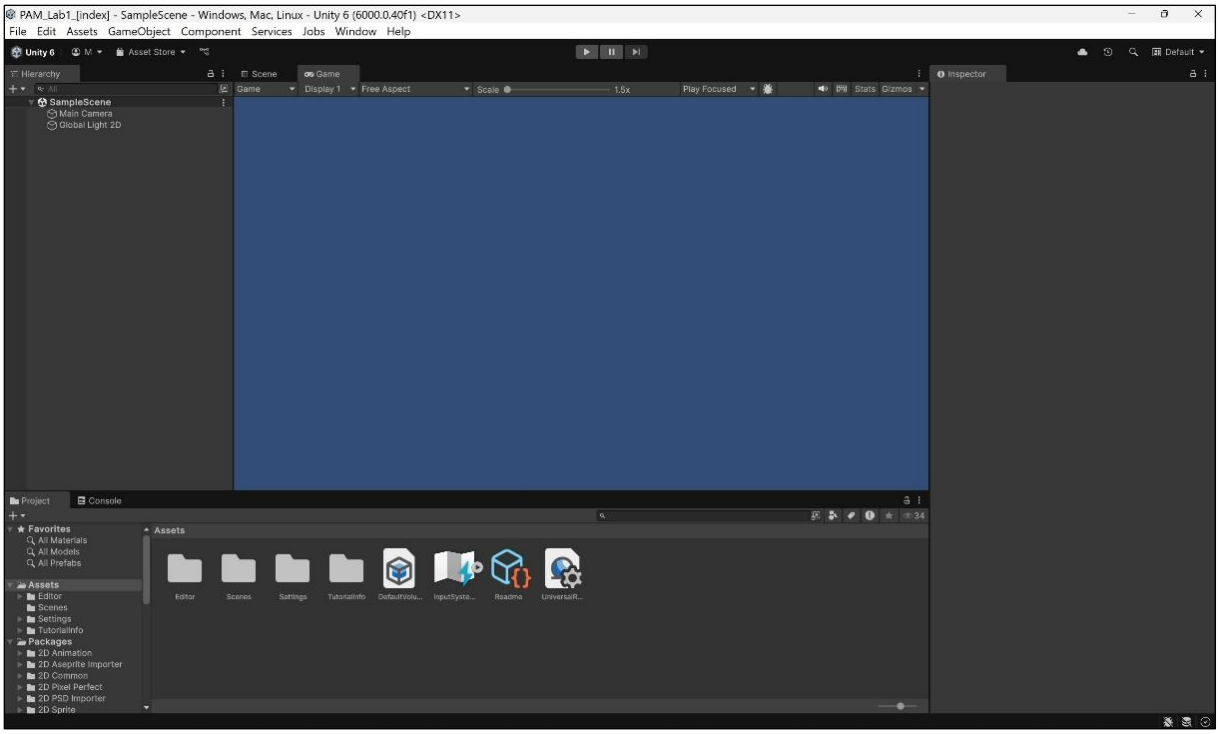
1. Hierarchii obiektów w scenie
2. Okna edycji sceny
3. Okno plików projektu / terminal
4. Inspektora właściwości wybranego obiektu



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.

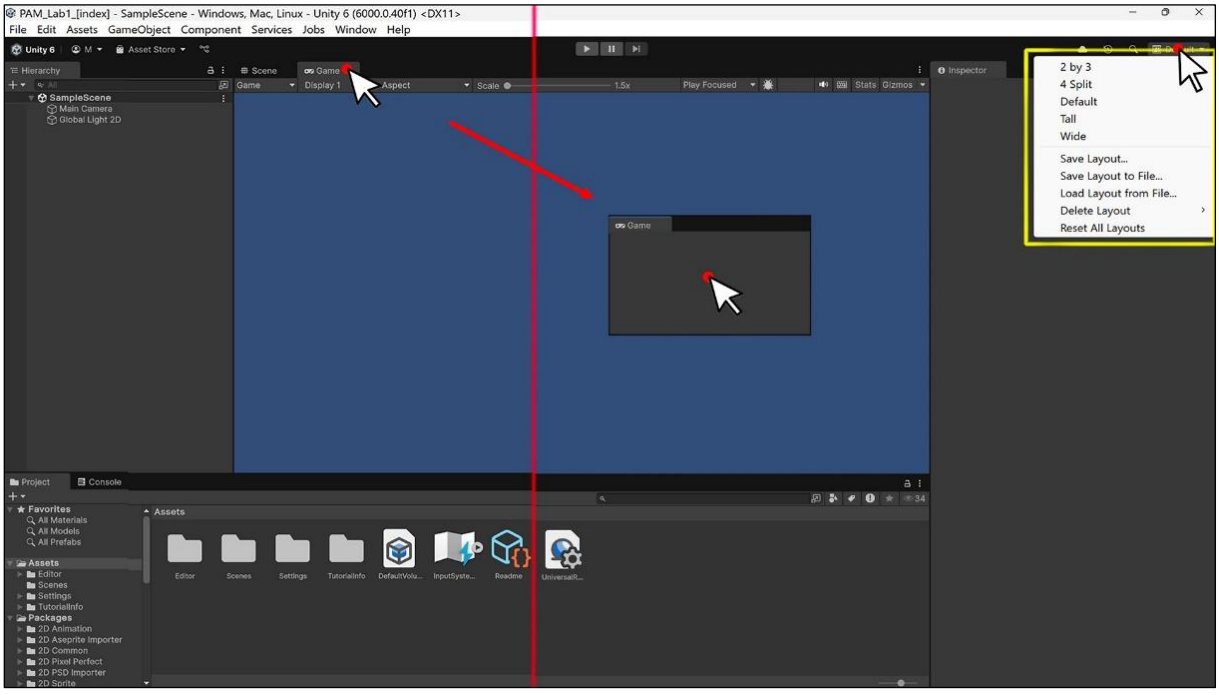
Należy mieć na uwadze, iż gdy aplikacja jest uruchomiona, to jakiekolwiek zmiany wprowadzone w tym czasie nie zostaną zapisane w projekcie!



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.

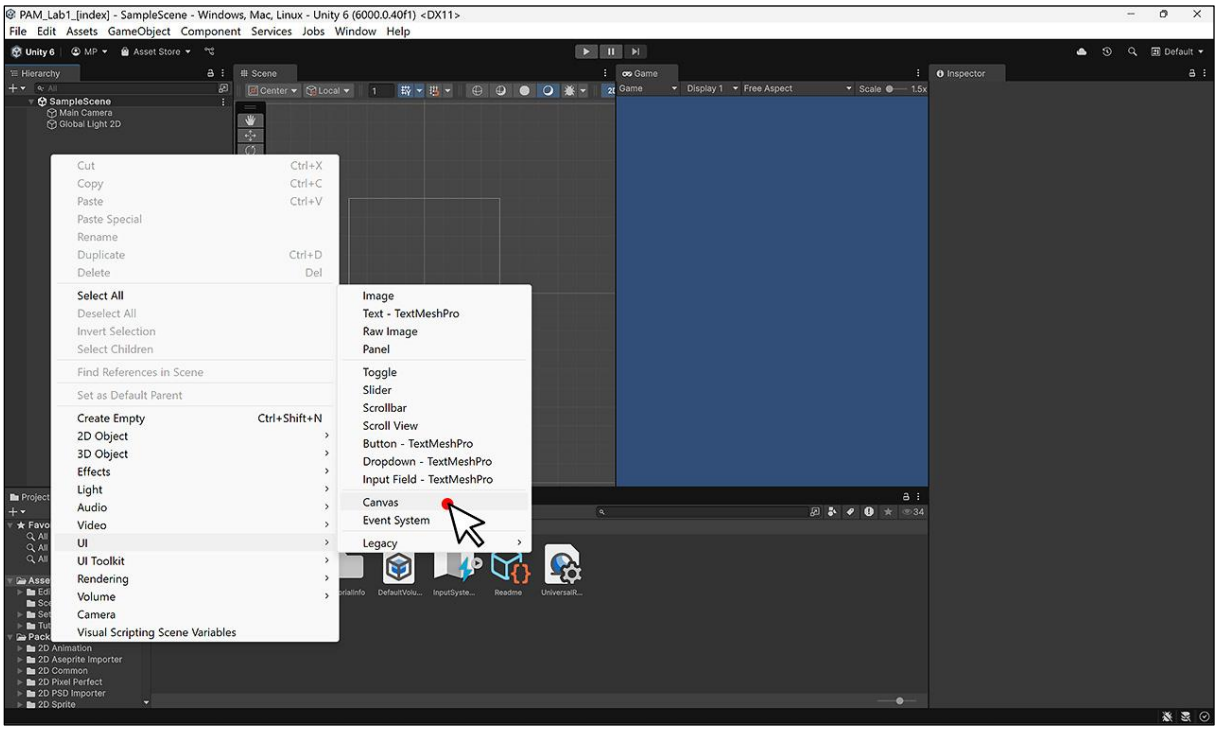
Istnieje również możliwość ręcznego przeciągnięcia okna GAME, tak aby pokazywało ono wygląd projektu. Aby przywrócić wygląd domyślny wystarczy nacisnąć przycisk **Layout** znajdujący się w prawym górnym rogu edytora i wybrać widok **Default**.



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**.

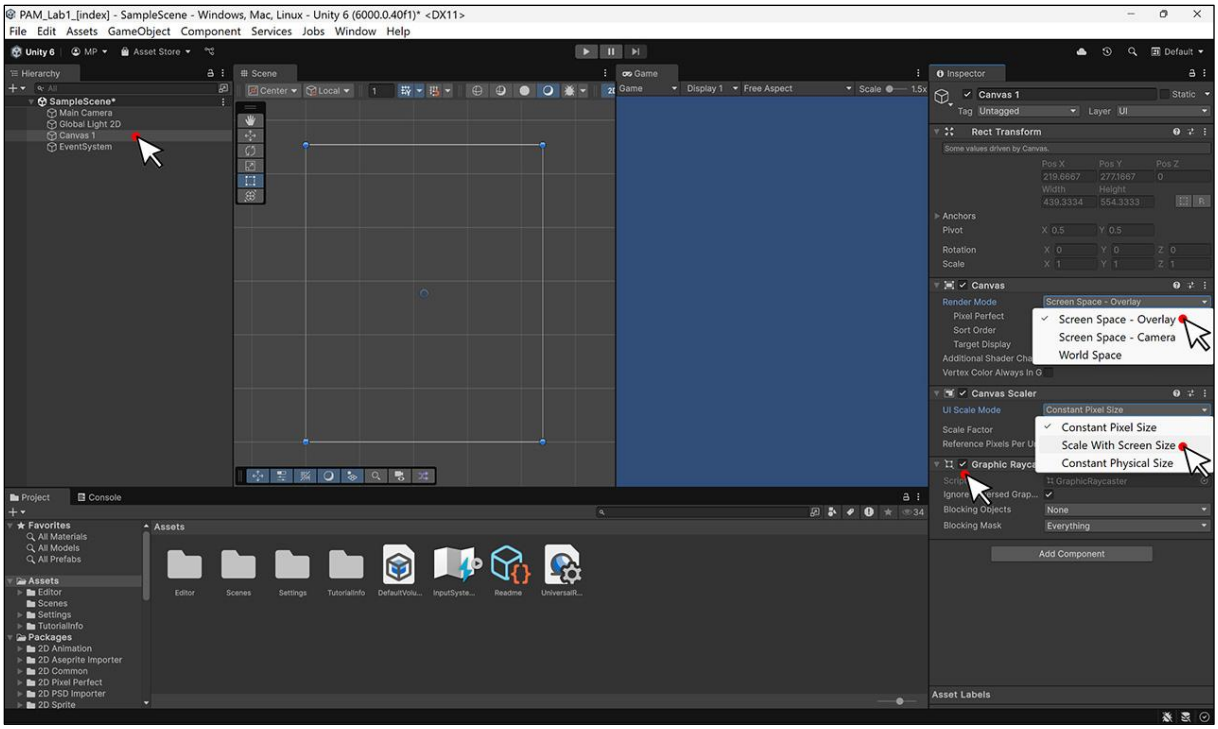


Rys. 15. Dodawanie obiektu Canvas

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 zaznaczmy 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.

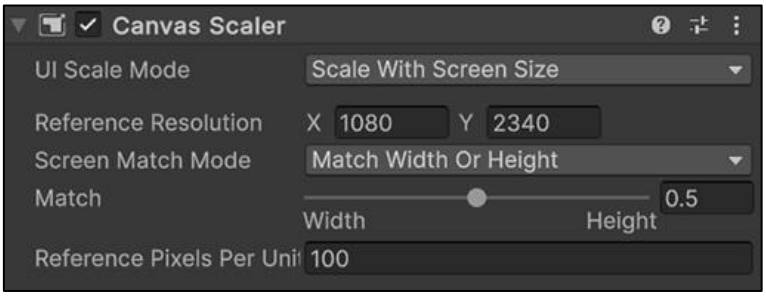
Zwracamy również uwagę na komponent o nazwie **Graphic Raycaster**. Określa on w jaki sposób obiekty wewnątrz naszego **Canvas** mają reagować na interakcje z myszą. Wyłączenie tego komponentu uniemożliwi wykrywanie takich interakcji, jednak oszczędzi zasoby obliczeniowe aplikacji, gdyż nie będzie ona 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.



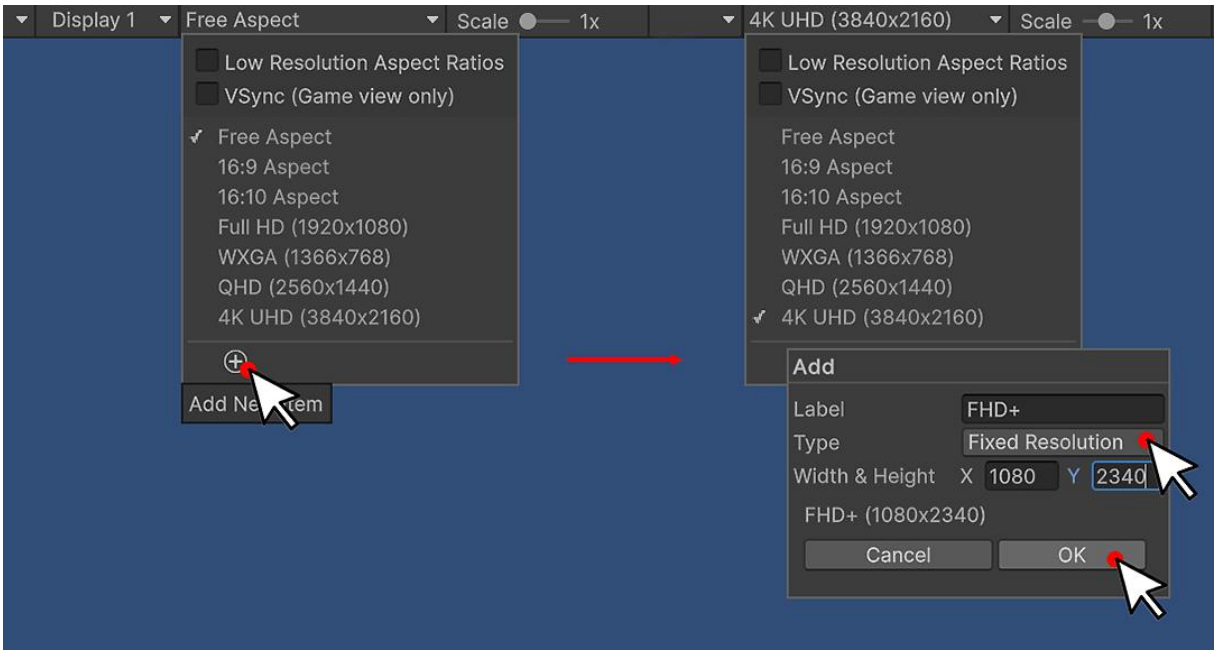
Rys. 16. Parametry obiektu Canvas

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 opcje **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

Określiliśmy 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 ostawiliśmy dla naszego obiektu. Zakładamy, iż będzie ona reprezentowała domyślny dla nas wyświetlacz. Zamiast rozdzielczości możemy określić też same proporcje ekranu - **Aspect Ratio**.

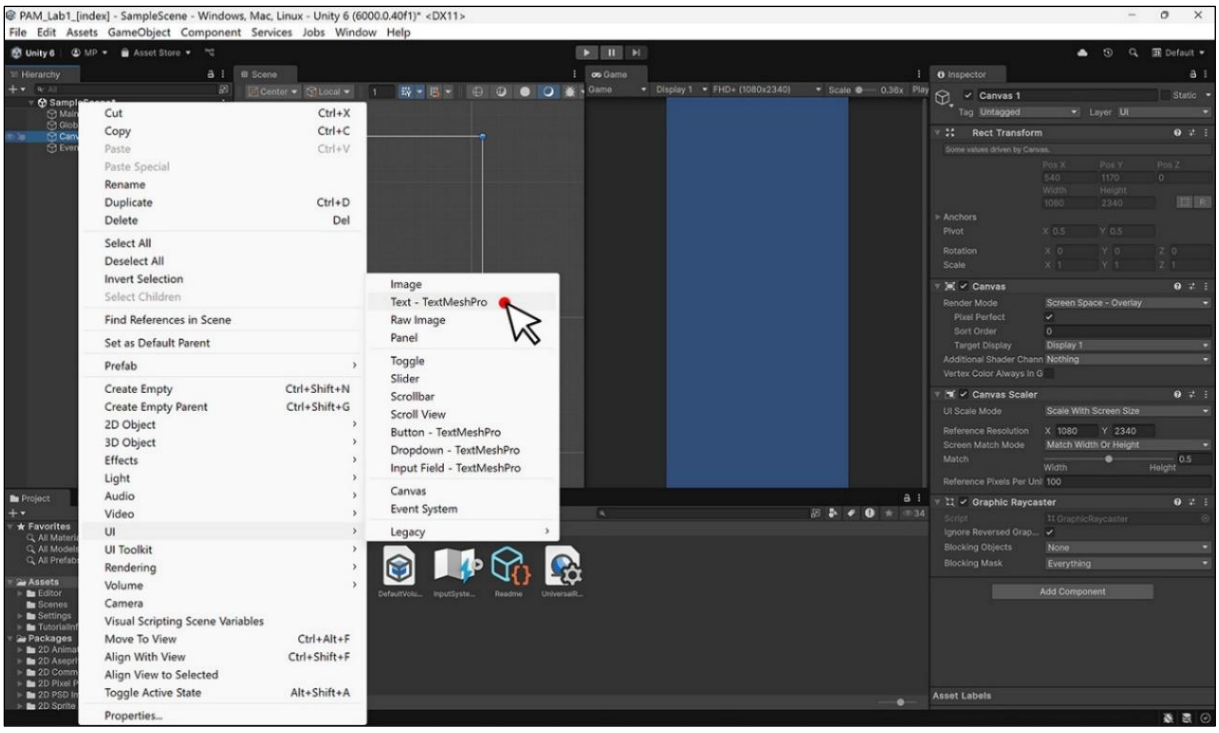


Rys. 18. Określenie wymiarów okna Game

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

5. Dodanie tła projektu

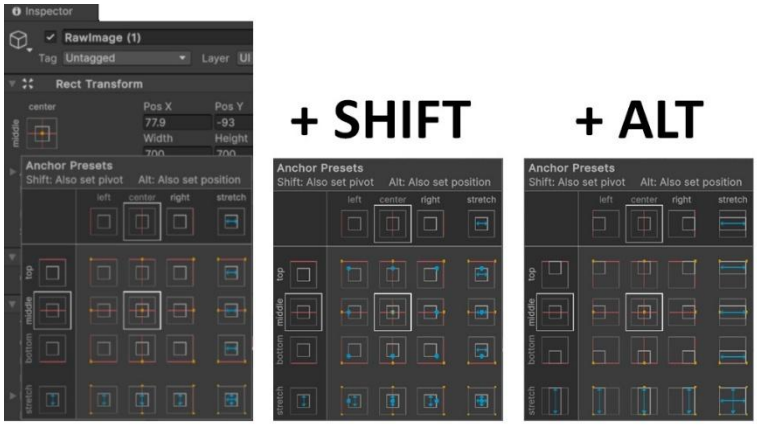
Dodajemy dwa obiekty typu **RawImage**. Wykonujemy to poprzez naciśnięcie prawym przyciskiem myszy na obiekt **Canvas 1** oraz wybór rodzaju obiektu z kategorii **UI**.



Rys. 19. Nowe obiekty GUI

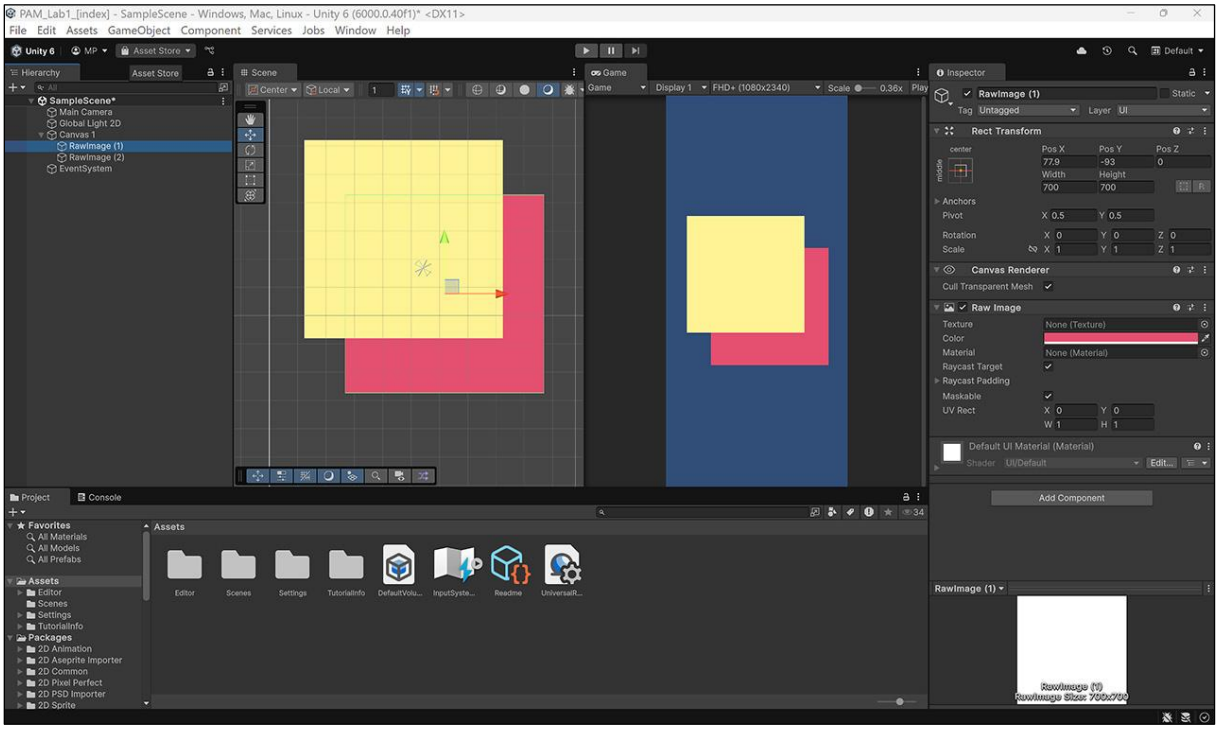
Podobnie jak w przypadku obiektu **Canvas**, tak i tu komponenty określają właściwości obiektów.

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)**.

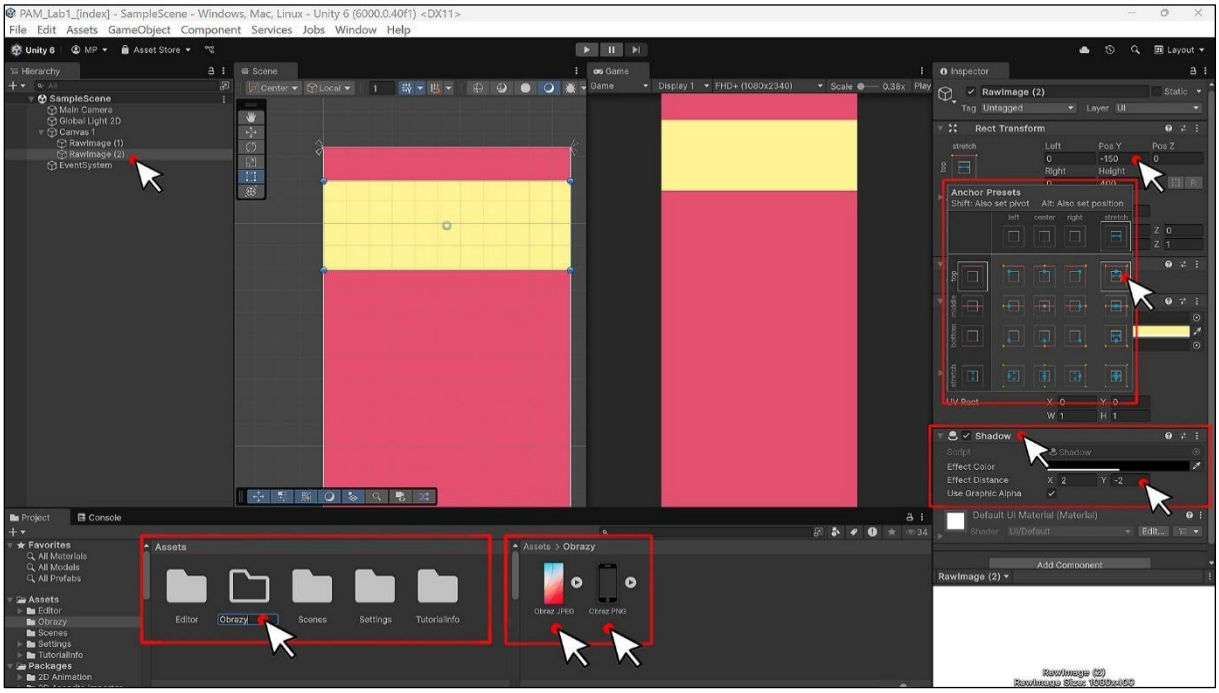


Rys. 21. Warstwy interfejsu

- Więcej na temat importu plików do projektu można przeczytać w artykule Importing Assets.
- Więcej na temat klas obiektów można przeczytać w artykule UnityEngine Objects.

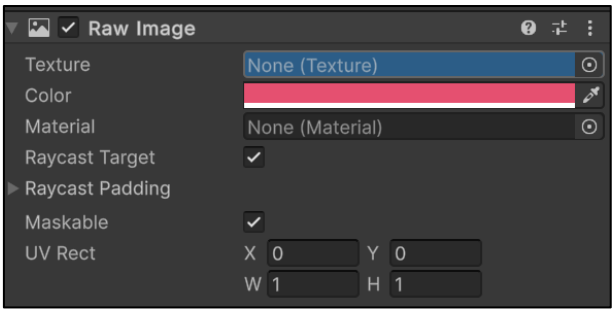
6. Tło nagłówka

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

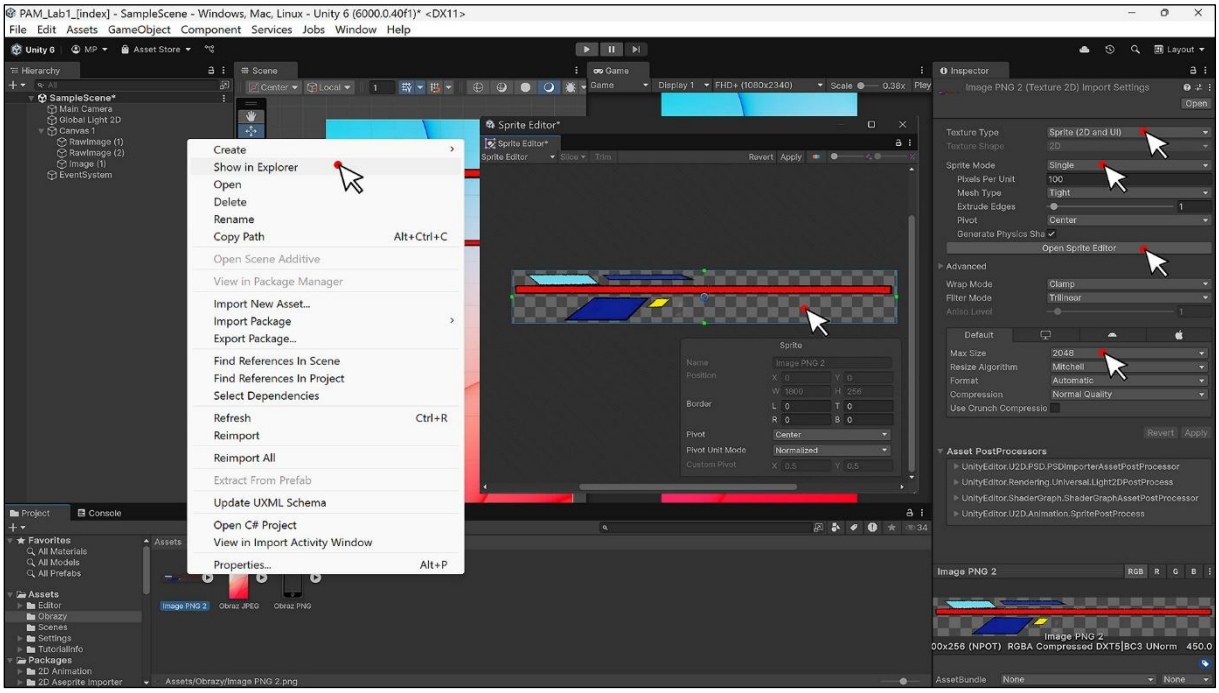
W celu urozmaicenia graficznego naszej aplikacji, tworzymy nowy folder o nazwie **Obrazy**. Pobieramy z Internetu dwie grafiki. Pierwsza będzie tłem, dlatego wystarczy nam obraz skompresowany **JPEG**. Drugi chcemy zastosować aby zaprezentować efekt przeźroczystości obrazu, dlatego konieczny będzie kanał 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**. Spróbujmy 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**.

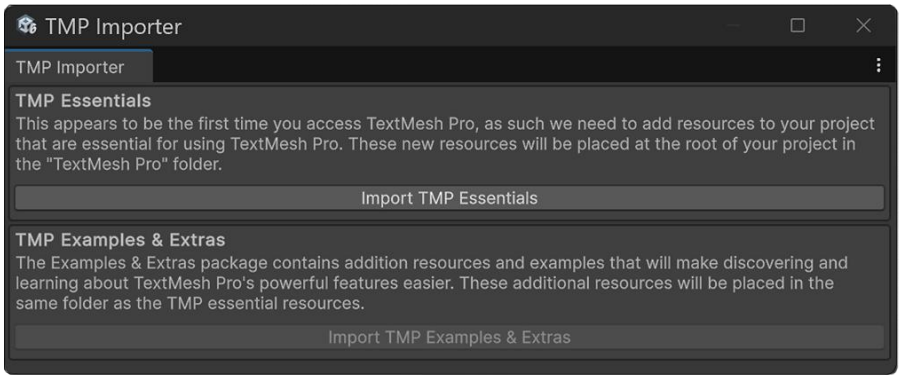


Rys. 24. Edycja ustawień obrazu

- Więcej na temat tekstur można znaleźć w artykule Get started with textures.
- Więcej na temat materiałów można znaleźć w artykule Materials.

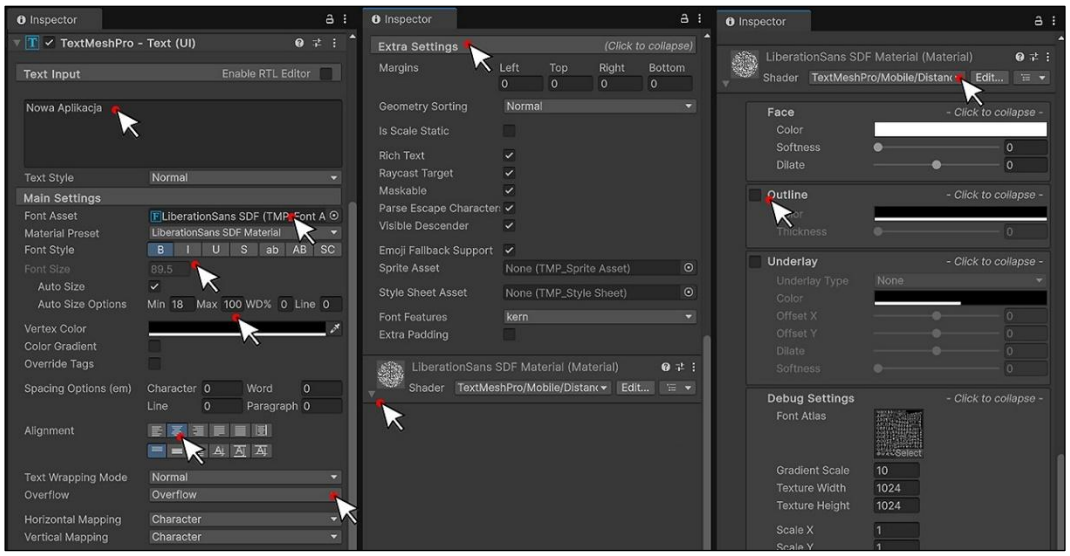
7. Tekst i napisy

Następnym krokiem jest dodanie tytułu naszej aplikacji. W tym celu dodajemy obiekt typu **Text – TextMeshPro**. TextMeshPro jest biblioteką odpowiadającą za zaawansowane wyświetlanie tektu. 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 dotyczyć tylko tego aktualnego obiektu tekstowego, jednak zmiany wprowadzone w ustawieniach materiału czcionki obejmą wszystkie obiekty używające tego samego materiału.

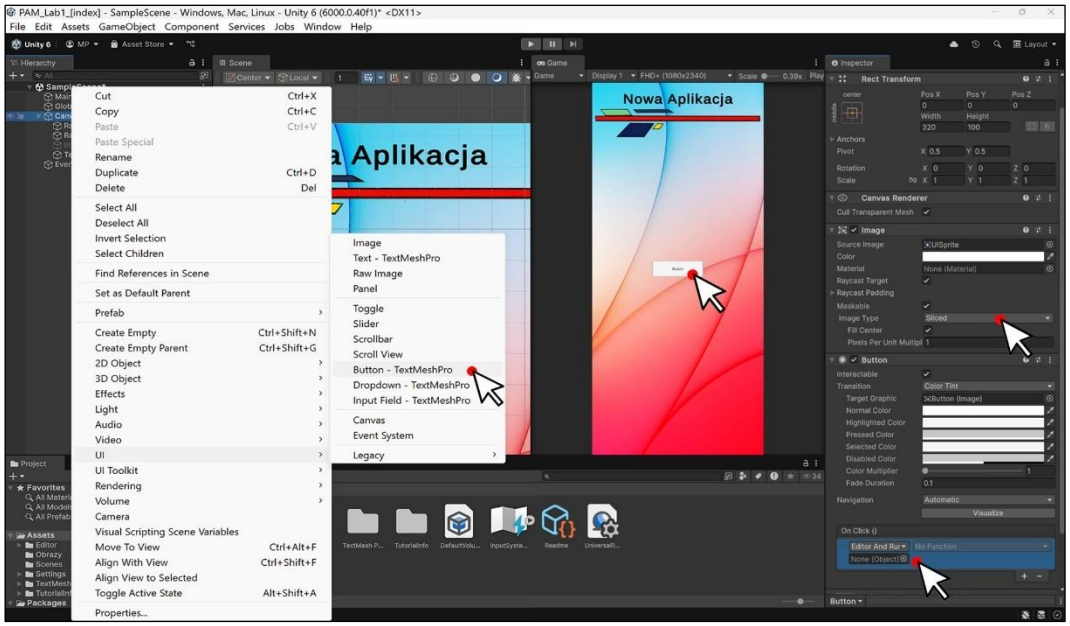


Rys. 26. Ustawienia obiektu tekstowego

- Więcej na ten temat można znaleźć w tym artykule - TextMesh Pro: Importing the Package.

8. Przyciski i interakcje

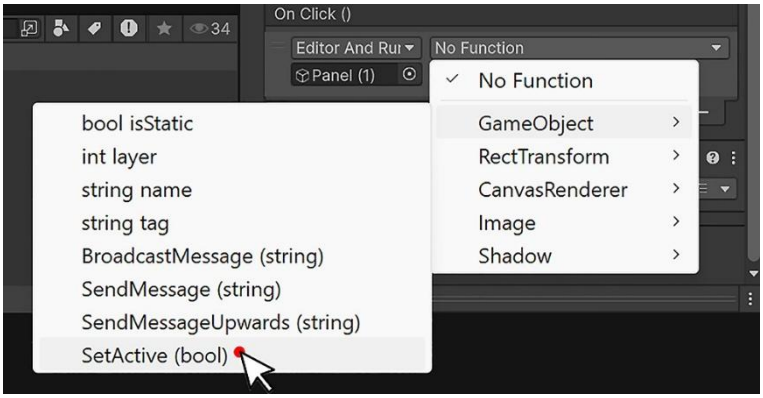
Kolejnym etapem jest dodanie przycisku do projektowanego ekranu, aby umożliwić użytkownikowi interakcję z aplikacją.



Rys. 27. Ustawienia przycisku

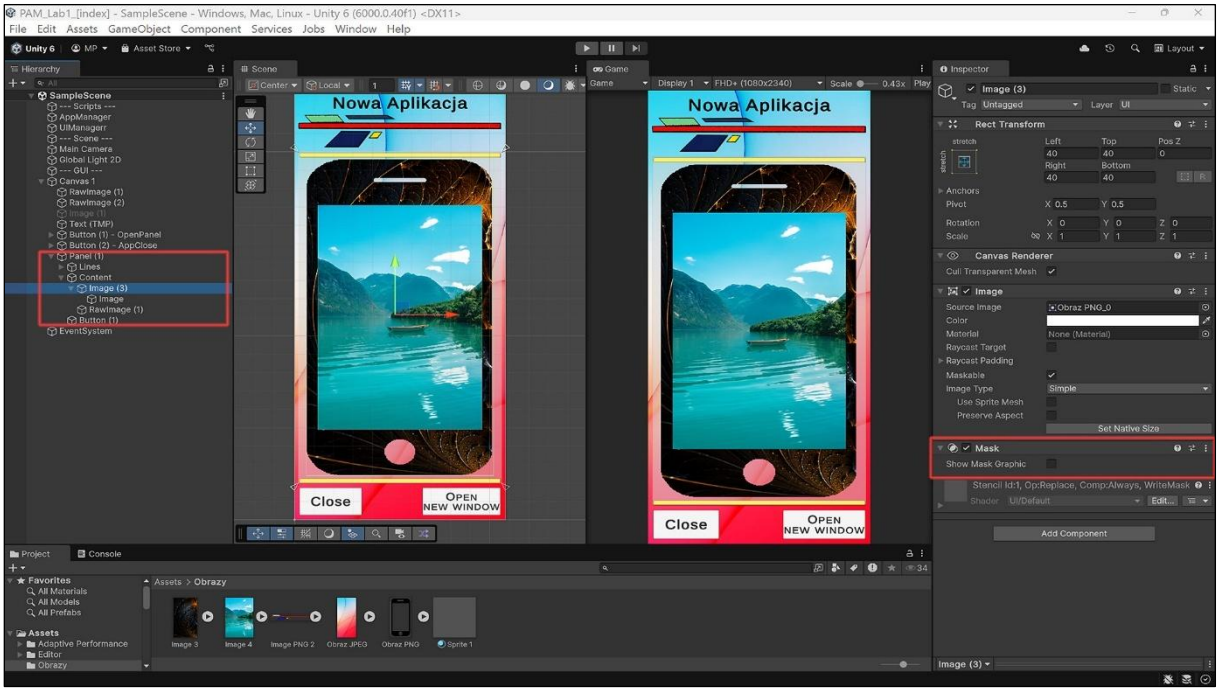
Główne ustawienia dotyczące tego jaką funkcję ma zrealizować naciśnięciu określamy w polu **On Click ()** znajdującego się na samym dole okna Inspektora. Z wybranej listy predefiniowanych funkcji możemy wpłynąć na dodany do tego pola obiekt lub jego komponenty. Dodajmy więc dwa

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 zmieńmy jego przeźroczystość 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.



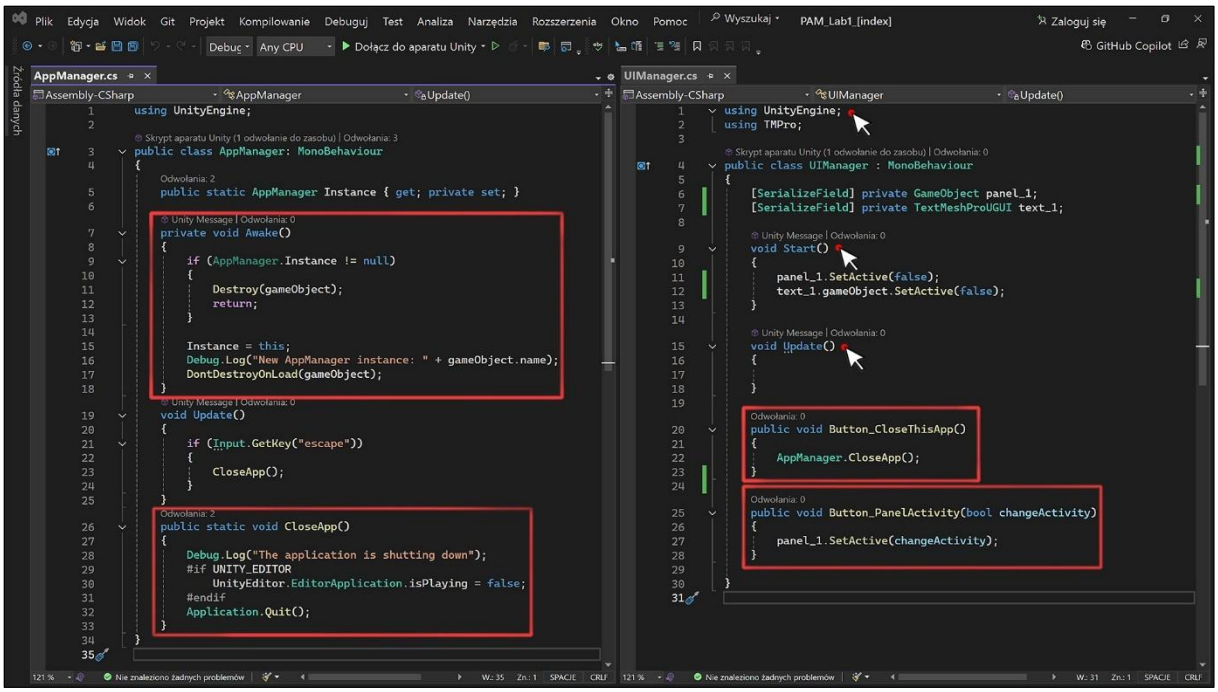
Rys. 29. Przykład użycia maski obrazu

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**. Wzorzec 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. Wzorzec 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). Dodatkowe 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

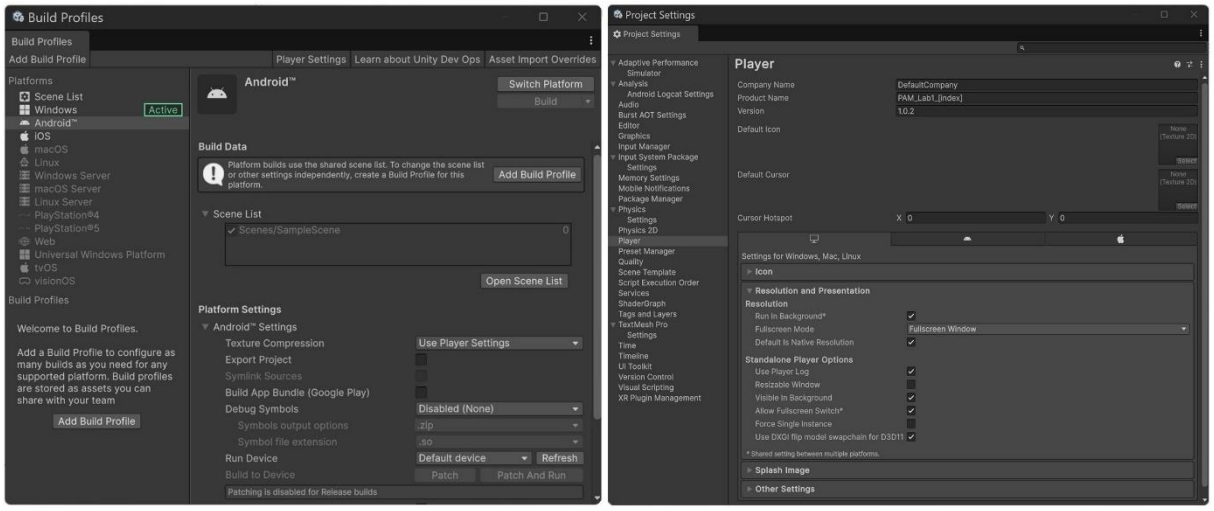
Istnieje jednak bardziej poprawny sposób na zarządzanie cyklem życia aplikacji mobilnej, w tym jej zamykania. Więcej na ten temat można znaleźć w artykule Quit a Unity Android application lub How do I programmatically quit my iOS application?

Czy można inaczej zaprojektować taką interakcje? 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,

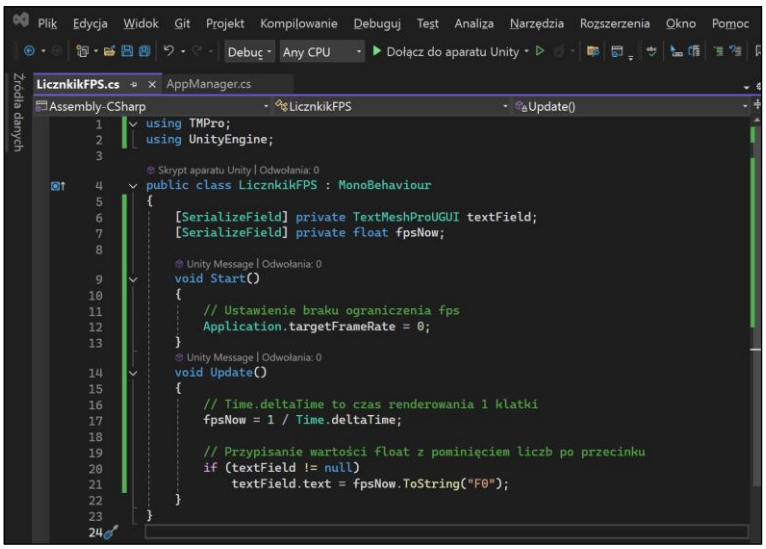


Rys. 31. Ustawienia projektu

- 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 dostępnie 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