Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Politechnika Świętokrzyska
Wydział Zarządzania i Modelowania Komputerowego

Kierunek studiów:
Inżynieria danych

Michał Pajęcki

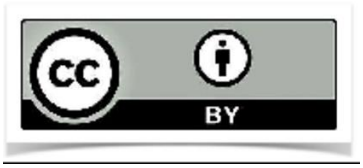
Materiały dydaktyczne do przedmiotu

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

opracowane w ramach realizacji Projektu
„**Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki**”
FERS.01.05-IP.08-0234/23

Kielce, 2025

Spis treści



Wstęp

W trakcie prac nad aktualizacją programu studiów dla kierunku ***inżynieria danych*** na Politechnice Świętokrzyskiej, prowadzonych we współpracy z praktykami z branży ICT, zwrócono uwagę na zmieniające się wymagania rynku pracy, a także trendy rozwoju sektora technologii informatycznych. Pracodawcy zgodnie podkreślali, że współczesny absolwent tego kierunku, oprócz bazowej znajomości klasycznych technik programowania, powinien także potrafić programować w środowiskach typu **no-code** / **low-code**, które coraz częściej wykorzystywane są w praktyce biznesowej oraz przemysłowej.

Wśród wielu przedmiotów informatycznych z którymi spotykają się Studenci kierunku ***inżynieria danych*** na Politechnice Świętokrzyskiej, na piątym semestrze studiów realizowany jest kurs pn. ***Programowanie urządzeń mobilnych***. W związku z powszechnością **urządzeń mobilnych** (np. telefonów komórkowych i tabletów) oraz ich niezwykle istotnej roli w życiu codziennym, umiejętność tworzenia aplikacji na takie urządzenia wydaje się być jednym z kluczowych obszarów współczesnych technologii informatycznych. Kompetencje w zakresie tworzenia **aplikacji mobilnych** są wręcz niejednokrotnie niezbędne w wielu dziedzinach nauki, biznesu i przemysłu. Tradycyjnie, aby programować takie aplikacje, należy wykorzystać jeden z języków programowania wysokiego poziomu (np. Java lub Kotlin – choć możliwości jest wiele). Technologie te wymagają nierzadko specjalistycznych, kierunkowych kompetencji programistycznych związanych z różnymi aspektami programowania urządzeń mobilnych (np. struktura projektu, zarządzanie cyklem życia aplikacji, czy też interakcja z systemem mobilnym).

Można wręcz stwierdzić, że przyswojenie jedynie tradycyjnych technik programistycznych nie jest już wystarczające – opanowanie podejść typu **no-code** / **low-code** stanowi w dzisiejszych czasach wartość dodaną na rynku pracy. Mając na uwadze powyższe, w niniejszych materiałach dydaktycznych została omówiona i zaprezentowana możliwość tworzenia aplikacji mobilnych przy wykorzystaniu wybranego narzędzia typu **no-code** / **low-code**: **MIT App Inventor**. Stanowi to poszerzenie treści podstawowych realizowanych w trakcie trwania kursu ***Programowanie urządzeń mobilnych***, a duża liczba podejmowanych zagadnień stanowi dodatkowe rozszerzenie umiejętności, gdyż wykracza poza ramy tego kursu. Dzięki dość szczegółowemu omówieniu wybranego środowiska, Studenci nabędą niezbędną wiedzę pozwalającą na tworzenie własnych aplikacji.

1. Istota urządzeń mobilnych i programowania no-code / low-code

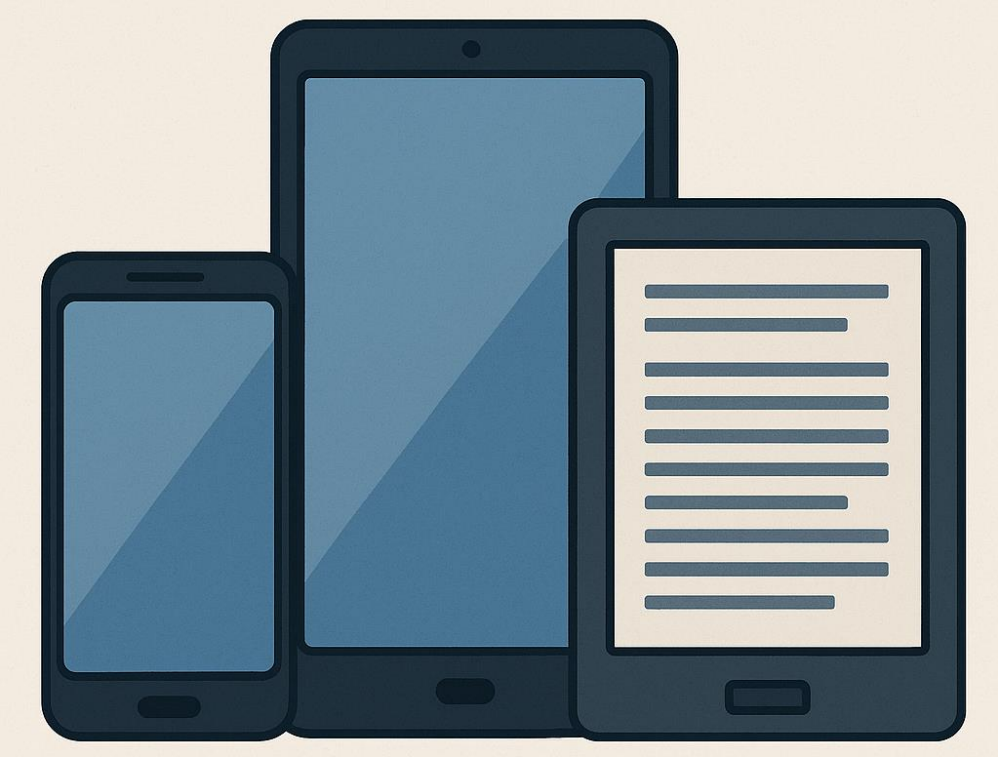
Dynamiczny rozwój urządzeń mobilnych zmienił sposób funkcjonowania współczesnego społeczeństwa, w którym w bardzo znaczącym stopniu wykorzystywane są różnorodne technologie mobilne. Smartfony, tablety i inne urządzenia przenośne stały się wręcz nieodłącznym elementem życia codziennego – zarówno w sferze prywatnej, jak i zawodowej, a także w trakcie nauki. Obecnie większość użytkowników korzysta z Internetu właśnie za pomocą urządzeń mobilnych. Znajomość podstaw programowania aplikacji na takie urządzenia przestaje być domeną wyłącznie „typowych" informatyków. W niniejszym rozdziale zostanie krótko nakreślona istota urządzeń mobilnych, natomiast bardziej zaawansowane informacje Student może odnaleźć w dostępnej literaturze. Zwrócono także uwagę, że zarówno dla potrzeb edukacyjnych, jak i biznesowego prototypowania aplikacji mobilnych, z powodzeniem można stosować rozwiązania typu **no-code** i **low-code**, takie jak np. **MIT App Inventor**, które pozwalają na szybkie tworzenie aplikacji w sposób wizualny, bez konieczności pisania kodu źródłowego w tradycyjny sposób; znajomość takich technik może być także przydatna w późniejszej pracy zawodowej.

1.1. Urządzenia mobilne – powszechność i definicja

Według badania Krajowego Instytutu Mediów (2025), w 2024 roku telefon komórkowy posiadało 97,5% polskich gospodarstw domowych (w tym 85,5% stanowiły smartfony), a tablet – 15,5% gospodarstw. Na 81% procentach tabletów zainstalowany był system operacyjny Android, a na 11,8% iOS. Warto wspomnieć, że posiadanie komputera stacjonarnego zadeklarowało jedynie 14% polskich gospodarstw domowych.

Urządzenie mobilne (ang. *mobile device*) można zdefiniować jako kompaktowe, przenośne urządzenie elektroniczne, które pozwala na przetwarzanie danych oraz komunikację bez potrzeby korzystania z przewodowego połączenia z Internetem. Posiada niewielkie rozmiary i z reguły zasilanie bateryjne, przez co może być swobodnie przenoszone przez jedną osobę bez konieczności stosowania dodatkowych akcesoriów transportowych. Współczesne urządzenia tego typu są najczęściej wyposażone w ekrany dotykowe i oferują rozbudowane możliwości łączności bezprzewodowej (np. Wi-Fi, sieć komórkowa, Bluetooth, NFC). Stanowią one dość zróżnicowaną grupę urządzeń i oferują szeroki zakres funkcjonalności i możliwości. Mogą różnić się od siebie wieloma cechami, m.in. parametrami ekranu,

procesorem, liczbą pamięci operacyjnej, wyposażeniem dodatkowym (np. akcelerometr, żyroskop, czujnik światła). Przykładami są **smartfony**, **tablety** i **czytniki e-booków** – rys. 1. Urządzenia mobilne mogą mieć różne kształty i rozmiary (“Mobile device“, 2025; Ross, Pillitteri, 2024; Sheldon, 2024; Stasiewicz, 2016).



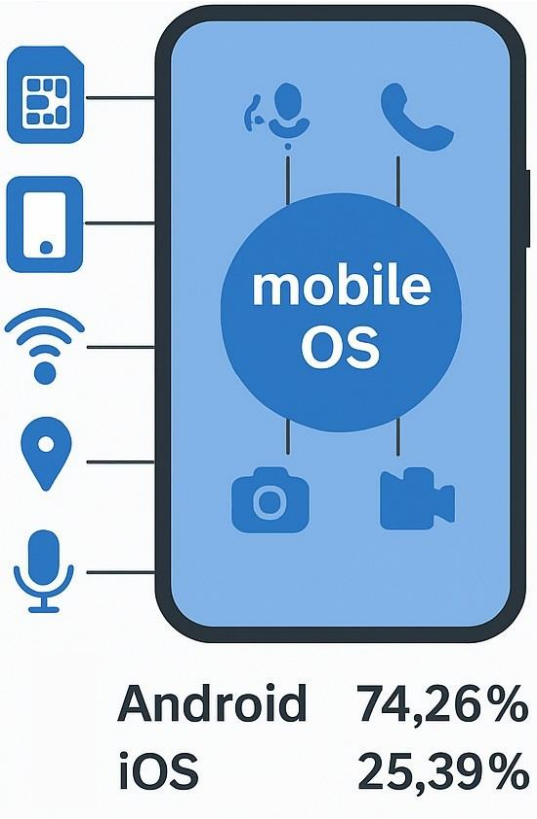
Rys. 1. Przykładowe urządzenia mobilne (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

[Tekst alternatywny. Rysunek 1 przedstawia trzy przykładowe urządzenia mobilne: smartfon, tablet oraz czytnik e-booków. Urządzenia różnią się rozmiarem i proporcjami ekranów. Jest to rysunek poglądowy.]

1.2. Mobilne systemy operacyjne i aplikacje

Podobnie jak komputery stacjonarne i laptopy, urządzenia mobilne również wymagają systemu operacyjnego – działają one pod kontrolą tzw. **mobilnego systemu operacyjnego** (ang. *mobile operating system*) – rys. 2, który zarządza zasobami sprzętowymi i programowymi. System taki łączy w sobie funkcjonalność klasycznych systemów operacyjnych oraz cech istotnych z punktu widzenia urządzeń przenośnych, które obejmują m.in. obsługę kart SIM, ekranów dotykowych, połączeń głosowych, komunikacji bezprzewodowej, systemów lokalizacji (GPS), a także

komponentów takich jak m.in. aparat fotograficzny, kamera, mikrofon, czujniki ruchu, dyktafon oraz odtwarzacz multimedialny. Systemy te powinny być ponadto zoptymalizowane pod kątem energooszczędności, mobilności i wygody użytkowania. W dzisiejszych czasach na rynku dominują dwa mobilne systemy operacyjne – **Android** i **iOS**. Należy mieć jednak świadomość, że w przeszłości było ich więcej – np. Symbian, czy też Windows Phone (“Mobile operating system“, 2025; Sheldon, 2024). Według serwisu StatCounter (2025), w czerwcu 2025 roku, w skali światowej, system Android był zainstalowany na 74,26% urządzeń mobilnych, natomiast iOS na 25,39%.



Rys. 2. Mobilny system operacyjny (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

[Tekst alternatywny. Rysunek 2 przedstawia smartfon z napisem "mobile OS" na ekranie, a wokół ikony reprezentujące funkcje systemów mobilnych: klawiatura, ekran dotykowy, sieć Wi-Fi, lokalizacja GPS, aparat fotograficzny, kamera, mikrofon oraz rozmowy głosowe. Na dole widnieją statystyki udziału w rynku mobilnych systemów operacyjnych: Android – 74,26%, iOS – 25,39%. Jest to rysunek poglądowy.]

Jak wspomniano powyżej, **Android** to obecnie najpopularniejszy system operacyjny dedykowany urządzeniom mobilnym. Pierwszą, finalną wersję Androida (1.0)

udostępniono we wrześniu 2008 roku. Wyposażony jest on w wiele gotowych do użycia funkcji i mechanizmów, które wspierają zarówno codzienne użytkowanie, jak i rozwój aplikacji. Stanowi pełnoprawną platformę programistyczną o otwartym kodzie źródłowym, bazującą na jądrze Linuksa, a jego rozwój jest nadzorowany i wspierany przez Google. Warto jednak zauważyć, że Android nie ogranicza się wyłącznie do smartfonów i tabletów. System ten rozwijany jest także w wersjach przystosowanych do innych typów urządzeń, m.in. telewizorów (Android TV), czy też samochodów (Android Auto) (“Android (operating system)“, 2025; Stasiewicz, 2016).

Standardowa aplikacja na system Android zbudowana jest z jednego lub większej liczby **ekranów**; każdy z nich oparty jest na tzw. **układzie** (ang. *layout*). Układ ten opisuje wygląd interfejsu użytkownika i najczęściej definiowany jest przy użyciu języka XML. Sam układ odpowiada jednakże tylko i wyłącznie za warstwę wizualną (obrazowo mówiąc: to, co widzi użytkownik). Aby aplikacja mogła reagować na działania użytkownika i wykonywać konkretne funkcje, konieczne jest utworzenie odpowiedniego kodu w wybranym języku programowania, takim jak np. Java lub Kotlin. Oficjalnym środowiskiem programistycznym IDE pozwalającym na tworzenie aplikacji mobilnych na system operacyjny Android jest **Android Studio** (“Android Developers“, 2025; Griffiths, Griffiths, 2018; Stasiewicz, 2016). Najważniejsze zagadnienia związane z programowaniem urządzeń mobilnych na Androida można znaleźć na oficjalnej stronie internetowej https://developer.android.com/get-started/overview.

iOS to mobilny system operacyjny (drugi najpopularniejszy na świecie pod względem liczby użytkowników) stworzony na urządzenia firmy Apple Inc. – smartfony iPhone. Stanowi także fundament innych systemów operacyjnych firmy Apple, m.in. iPadOS dla tabletów, tvOS dla Apple TV oraz watchOS dla inteligentnych zegarków Apple Watch. Wywodzi się on z systemu operacyjnego macOS. Pierwsza wersja została zaprezentowana 29 czerwca 2007 roku pod nazwą iPhoneOS 1.0 (“iOS“, 2025). Na obecną chwilę preferowanym przez Apple językiem do programowania aplikacji na system iOS jest Swift, jednak wiele aplikacji jest zbudowanych wykorzystując Objective-C (“Apple Developer“, 2025). Najważniejsze zagadnienia związane z programowaniem urządzeń mobilnych na iOS można znaleźć na oficjalnej stronie internetowej https://developer.apple.com/learn/.

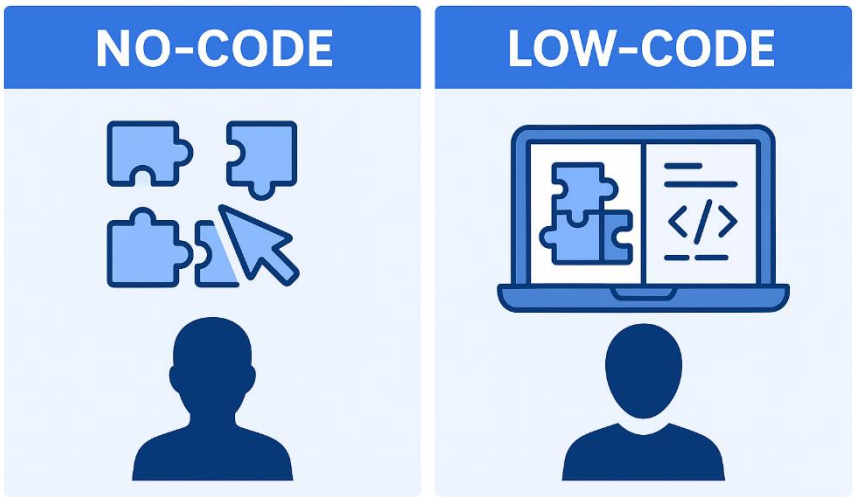
Aplikacja mobilna (ang. *mobile application*) to oprogramowanie przeznaczone do działania na urządzeniu mobilnym. Aplikacje takie można podzielić na **natywne** (tworzone specjalnie dla danego systemu operacyjnego, np. Android lub iOS), **internetowe** (działające najczęściej w przeglądarce internetowej) oraz **hybrydowe** (łączące cechy dwóch wcześniejszych) (“Mobile app“, 2025). Według *Ustawy z dnia*

4 kwietnia 2019 r. o dostępności cyfrowej stron internetowych i aplikacji mobilnych podmiotów publicznych (Dz.U. 2019 poz. 848), jest to „publicznie dostępne oprogramowanie z interfejsem dotykowym zaprojektowane do wykorzystania na przenośnych urządzeniach elektronicznych, z wyłączeniem aplikacji przeznaczonych do użytku na przenośnych komputerach osobistych".

W niniejszej publikacji nie będą poruszane zagadnienia dotyczące tworzenia aplikacji w dedykowanych dla danej platformy językach programowania wysokiego poziomu (np. dla Androida – Kotlin i Java, a dla iOS – Objective-C i Swift).

1.3. Narzędzia typu no-code / low-code

W codziennym życiu pojęcia **no-code** i **low-code** bywają traktowane jako synonimy. Oba te podejścia pozwalają na tworzenie aplikacji bez konieczności pisania kodu w tradycyjny sposób (całkowicie lub w znacznym stopniu), co bez wątpienia pozwala znacząco skrócić czas tworzenia oprogramowania. Najczęściej wykorzystywane jest kodowanie wizualne oparte na mechanizmie typu **„przeciągnij i upuść"** (ang. *drag and drop*). W konsekwencji, możliwość tworzenia aplikacji dostępna jest dla szerszego grona użytkowników, nawet bez formalnego wykształcenia informatycznego. Można jednak wypunktować pewne różnice (rys. 3), szczególnie w zakresie grupy docelowej użytkowników oraz wejściowych wymagań dotyczących wiedzy programistycznej. Oba wspomniane podejścia są w dzisiejszych czasach szeroko wykorzystywane przez przedsiębiorstwa w celu szybkiego tworzenia różnorodnych aplikacji biznesowych, dostosowanych do dynamicznie zmieniających się potrzeb (Kostereva, Kawasaki, 2022).



Rys. 3. No-code vs low-code (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

[Tekst alternatywny. Rysunek 3 przedstawia porównanie podejść no-code i low-code. W platformach no-code użytkownik tworzy aplikację za pomocą bloków, bez potrzeby pisania kodu. W podejściu low-code dodawane są także fragmenty kodu, co pozwala na większą elastyczność i rozbudowę aplikacji. Jest to rysunek poglądowy.]

Platformy typu **no-code** są przeznaczone do użytku także przez osoby niebędące programistami (np. analitycy, czy też pracownicy działów biznesowych), co oznacza, że nacisk kładziony jest w nich na wyeliminowanie potrzeby „ręcznego" pisania kodu w trakcie tworzenia aplikacji. Oczywiście, podejście to wymaga od użytkownika rozumienia procesów biznesowych, myślenia analitycznego i umiejętności twórczego rozwiązywania problemów. Musi on także nauczyć się obsługi wybranej platformy **no-code**, jednak nie musi poznawać żadnego konkretnego języka programowania. Tak jak wspomniano, programowanie odbywa się w formie wizualnej (graficznej), najczęściej wykorzystując gotowe komponenty i metodę typu „przeciągnij i upuść". Prototypowanie aplikacji jest więc bardzo szybkie, a wymagania wejściowe w zakresie umiejętności programistycznych użytkownika są niskie. Możliwość tworzenia logiki aplikacji jest jednakże ograniczona tylko i wyłącznie do istniejących komponentów, co oznacza małą elastyczność. W platformach typu **low-code** nacisk kładziony jest z kolei jest na znaczne zmniejszenie ilości kodu, który musi zostać napisany przez programistę, jednak nie eliminuje się go całkowicie. Przez to nawet mniej doświadczeni programiści są w stanie przyspieszyć proces tworzenia aplikacji dla różnorodnych zastosowań. Większość kodu tworzona jest w formie wizualnej, jednakże można dodawać fragmenty kodu w celu rozbudowania funkcjonalności oprogramowania ("Low-code development platform"; 2025; "No-code development platform", 2025; Kostereva, Kawasaki, 2022).

Istnieje wiele narzędzi **no-code** / **low-code** pozwalających na tworzenie aplikacji mobilnych, m.in. **MIT App Inventor**, **Thunkable**, **Kodular**, **Adalo**, **FlutterFlow**. Różnią się one między sobą zakresem dostępnych funkcjonalności, poziomem zaawansowania, a także możliwościami integracji z innymi usługami. Należy również zaznaczyć, że nie wszystkie z nich są całkowicie darmowe, gdyż część z nich oferuje jedynie ograniczone plany bezpłatne; bardziej zaawansowane funkcje dostępne są z kolei w wersjach płatnych.

2. Projektowanie aplikacji mobilnych z wykorzystaniem MIT App Inventor

W niniejszym opracowaniu, do tworzenia aplikacji mobilnych, zostanie wykorzystane narzędzie dostępne całkowicie za darmo – **MIT App Inventor**. Dzięki swojej intuicyjności, bogatej palecie komponentów oraz możliwości pracy w przeglądarce internetowej doskonale sprawdza się zarówno w edukacji, jak i przy realizacji rzeczywistych projektów.

Podczas tworzenia kolejnych podrozdziałów bazowano na oficjalnej dokumentacji narzędzia MIT App Inventor (“Reference Documentation“, 2025; “The MIT App Inventor Library: Documentation & Support“, 2025) oraz społeczności internetowej tego narzędzia (“MIT App Inventor Community“, 2025). Wykorzystano także źródła (“MIT App Inventor“, 2025; “App Inventor“, 2024).

2.1. Podstawowe informacje o MIT App Inventor

MIT App Inventor stanowi otwartoźródłowe środowisko programistyczne, początkowo utworzone przez firmę Google, a obecnie rozwijane i utrzymywane przez zespół MIT (*Massachusetts Institute of Technology*). Z punktu widzenia procesu dydaktyki, jak i późniejszej użyteczności tworzonych aplikacji, rozwiązanie to wydaje się być optymalne. Co więcej, wszystkie funkcjonalności dostępne są całkowicie bezpłatnie. Jest to narzędzie działające online (dostępne przez przeglądarkę internetową), a wszystkie projekty przechowywane są w chmurze. Do tworzenia aplikacji mobilnych wystarczy więc wspomniana przed chwilą przeglądarka internetowa i bezpłatne konto Google – chociaż istnieje możliwość tworzenia aplikacji bez jego posiadania. Wspomniane cechy pozwalają więc Studentowi na bezproblemową pracę nad swoimi projektami z dowolnego miejsca i urządzenia, bez konieczności instalowania dodatkowego oprogramowania; aczkolwiek oczywiście wymagany jest dostęp do Internetu. Tworzone aplikacje można testować na swoim fizycznym urządzeniu w czasie rzeczywistym (zarówno wyposażonym w system Android, jak i iOS) z wykorzystaniem dedykowanego oprogramowania. Co więcej, wcale nie trzeba posiadać fizycznego urządzenia mobilnego, gdyż jest także możliwość wykorzystania emulatora telefonu komórkowego, który instalowany jest bezpośrednio na komputerze. W ogólności, na obecną chwilę, istnieje pełne wsparcie dla aplikacji tworzonych na system operacyjny Android. Warto podkreślić, że możliwe jest nawet ich kompilowanie do pliku instalacyjnego ***.apk**, który następnie można zainstalować bezpośrednio na fizycznym urządzeniu; co więcej, mogą być one także publikowane w Sklepie Play. Z kolei możliwość tworzenia aplikacji na system iOS jest

ciągle rozwijana i wszystkie funkcjonalności nie są obecnie wspierane, a o postępach prac można czytać bezpośrednio na stronie https://appinventor.mit.edu/ios_tips.

MIT App Inventor to dość rozbudowane, a z drugiej strony jednocześnie intuicyjne narzędzie, które umożliwia tworzenie w pełni funkcjonalnych aplikacji mobilnych bez potrzeby pisania kodu źródłowego ani znajomości języków programowania, przez co może być traktowane jako narzędzie typu **no-code**. Programy tworzy się projektując wygląd interfejsu użytkownika, a następnie „programując" logikę działania aplikacji za pomocą gotowych komponentów i bloków kodu „tak jak puzzle", wykorzystując podejście „przeciągnij i upuść". Jeśli jednak użytkownik potrzebuje utworzyć bardziej zaawansowany projekt, można wykorzystać podejście **low-code** i skorzystać z rozszerzeń (ang. *extensions*), do czego wymagana jest znajomość języka Java, czy też integrować zewnętrzne API.

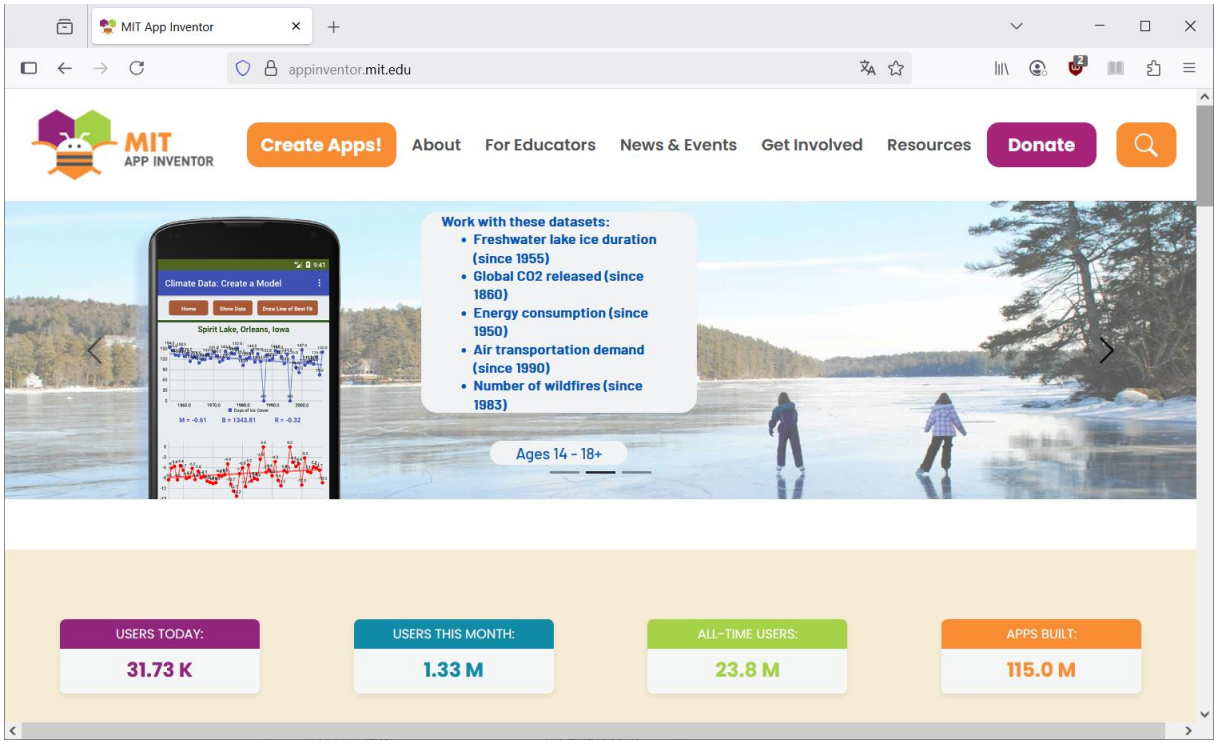
Platforma MIT App Inventor wydaje się być więc odpowiednia zarówno dla początkujących, nawet nietechnicznych użytkowników w procesie dydaktyki (uczniowie, studenci), jak i dla bardziej zaawansowanych osób (mniej lub bardziej doświadczonych programistów), którzy chcą szybko tworzyć własne projekty na urządzenia mobilne, mając do dyspozycji dość bogatą bibliotekę komponentów. Istnieje aktywna społeczność użytkowników, co może być pomocne w nauce i rozwiązywaniu ewentualnych problemów.

2.2. Logowanie i menu Projekty

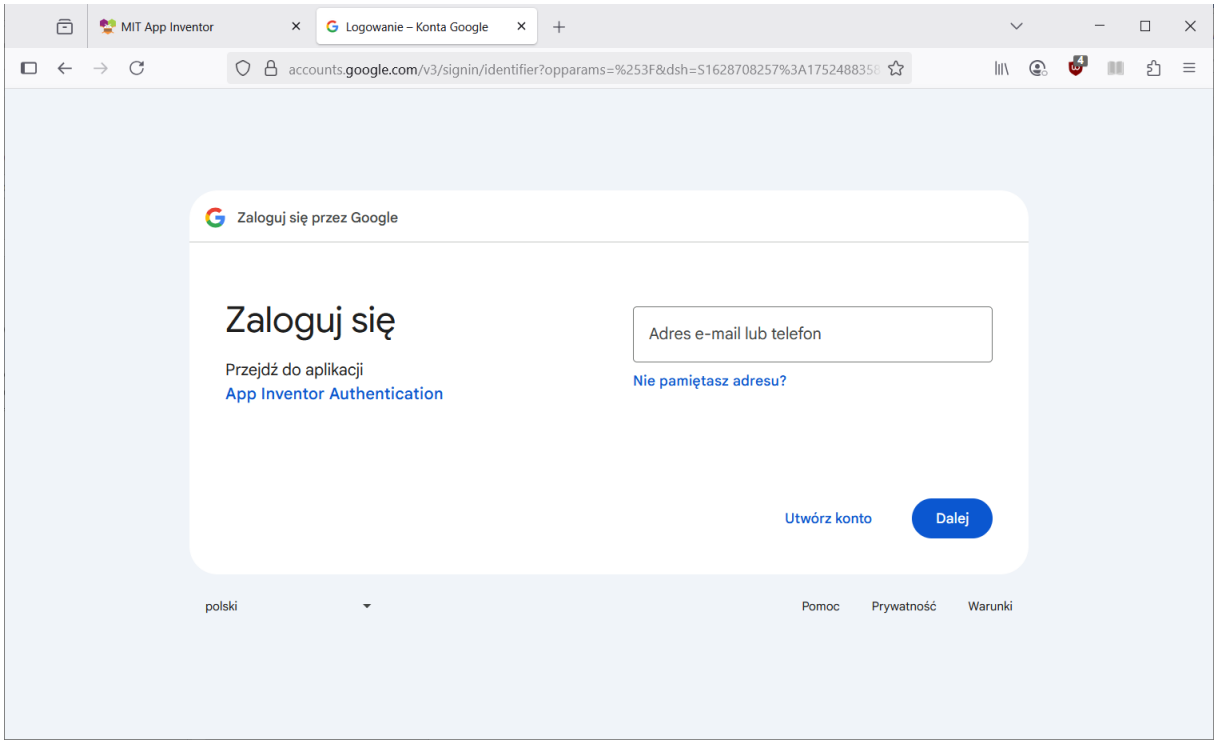
Aby otworzyć narzędzie MIT App Inventor należy uruchomić przeglądarkę internetową (np. Google Chrome lub Mozilla Firefox) i wejść na stronię internetową https://ai2.appinventor.mit.edu. Następnie należy kliknąć przycisk **Create Apps!** (rys. 4).

[Tekst alternatywny. Rysunek 4 przedstawia screen ekranu strony głównej MIT App Inventor o adresie https://ai2.appinventor.mit.edu otwartej w przeglądarce internetowej Mozilla Firefox. Na górze widać przycisk Create Apps! na pomarańczowym tle.]

W kolejnym kroku należy zalogować się na konto Google (rys. 5), przy wykorzystaniu którego będzie można pracować w narzędziu MIT App Inventor – wszystkie utworzone projekty przechowywane są w chmurze w powiązaniu z tym kontem.



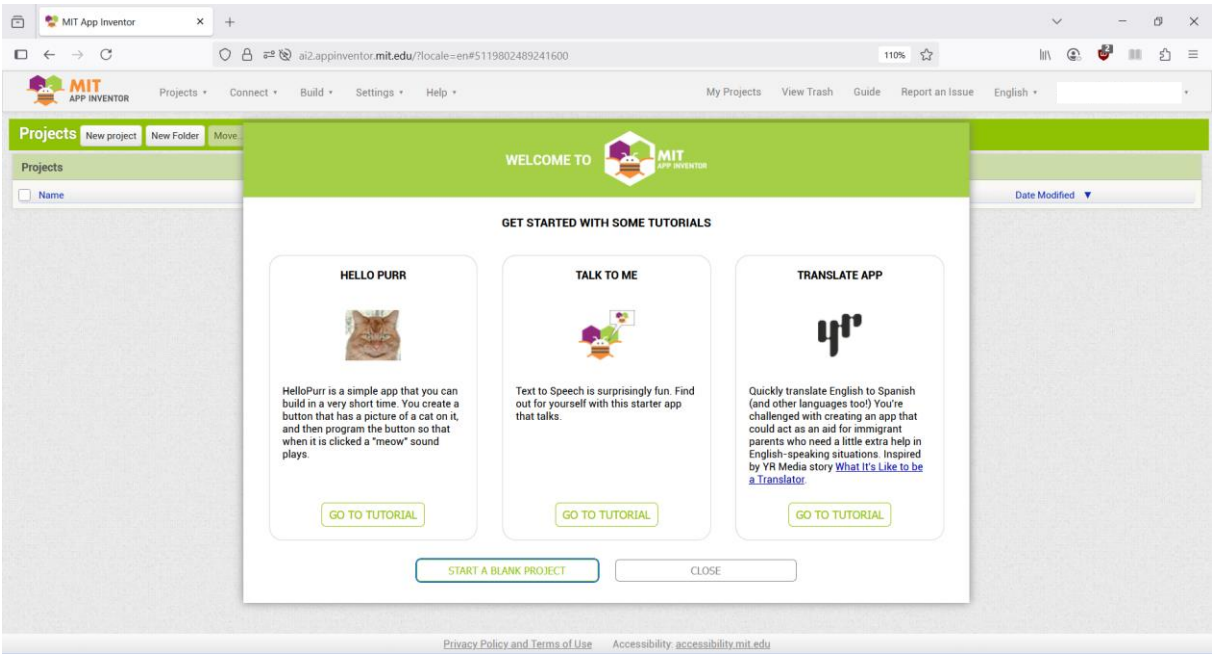
Rys. 4. Strona internetowa https://ai2.appinventor.mit.edu



Rys. 5. Logowanie do narzędzia MIT App Inventor przy wykorzystaniu konta Google

[Tekst alternatywny. Rysunek 5 przedstawia screen ekranu, na którym znajduje się formularz logowania do narzędzia MIT App Inventor przy wykorzystaniu konta Google. Należy wprowadzić adres e-mail lub telefon, a w kolejnym kroku hasło.]

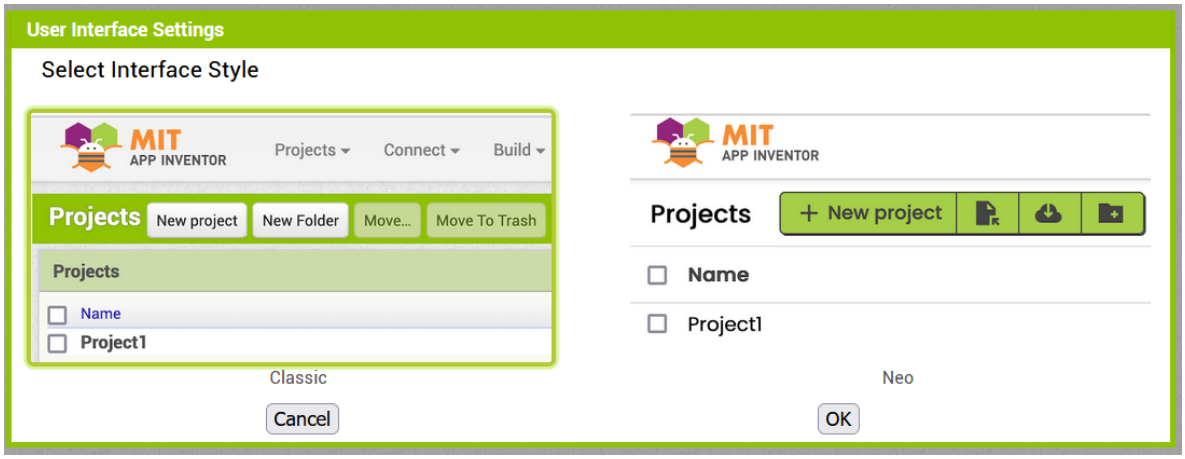
Po poprawnym zalogowaniu otwiera się projekt, nad którym ostatnio pracował użytkownik, a w przypadku braku jakichkolwiek projektu – okno powitalne (rys. 6). Po kliknięciu na **CLOSE** można zmienić język interfejsu narzędzia MIT App Inventor na polski – z rozwijalnej listy w górnej części narzędzia, obok słowa **English,** należy wybrać **Polski**; niestety, w chwili tworzenia niniejszych materiałów nie wszystkie teksty, polecenia i opisy zostały przetłumaczone – część z nich ciągle jest w języku angielskim – rys. 8.



Rys. 6. Okno powitalne narzędzia MIT App Inventor

[Tekst alternatywny. Rysunek 6 przedstawia screen ekranu, na którym znajduje się okno powitalne w narzędziu MIT App Inventor. Teksty pisane są w języku angielskim.]

Przy pierwszym uruchomieniu może także nastąpić pytanie o wybór stylu interfejsu użytkownika – rys. 7; w niniejszym opracowaniu wybrano styl **Classic**.

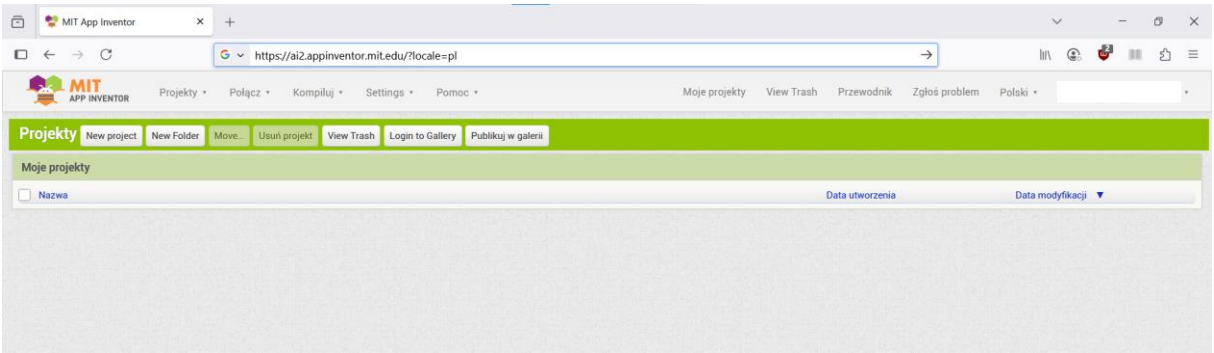


Rys. 7. Wybór stylu interfejsu użytkownika w narzędziu MIT App Inventor

[Tekst alternatywny. Rysunek 7 przedstawia fragment okna narzędzia MIT App Inventor, w którym można wybrać styl: Classic (ten jest zaznaczony) lub Neo.]

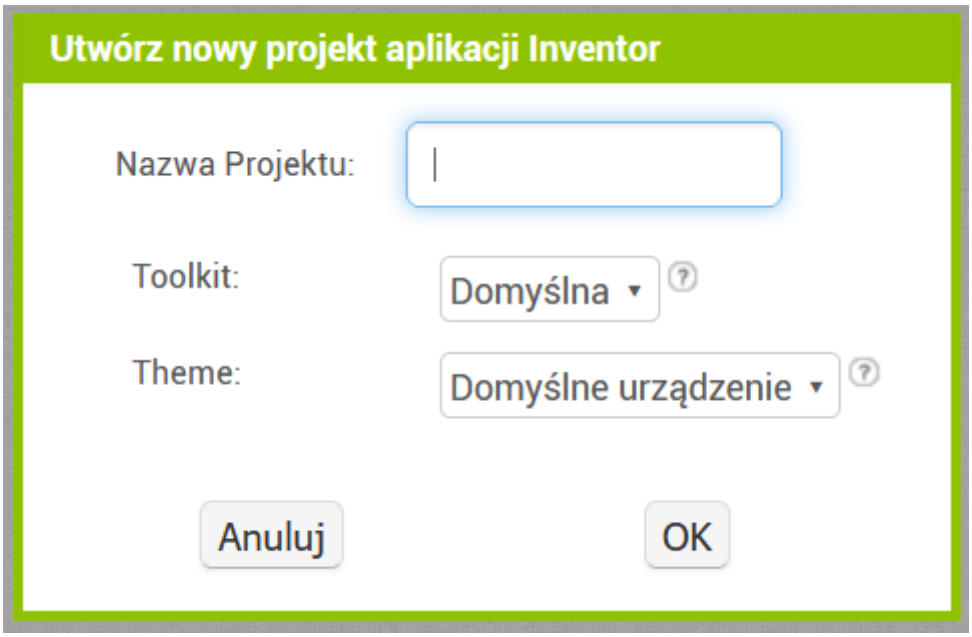
W razie konieczności, styl narzędzia MIT App Inventor można zmienić w dowolnej chwili wybierając z menu górnego **Settings → User Interface Settings**.

W zakładce **Projekty**, której górny fragment przedstawiony jest na rys. 8, przechowywane są wszystkie projekty aplikacji mobilnych utworzonych przez danego użytkownika. Aby utworzyć nowy projekt, należy wybrać polecenie **New project** – pojawi się nowe okno, w którym należy podać jego nazwę – rys. 9; musi ona rozpoczynać się od litery, a także może zawierać wyłącznie litery, cyfry i znaki podkreślenia. Polskie znaki nie są dopuszczalne. Można także wybrać zestaw komponentów potrzebnych do projektu (**Toolkit**) i motyw używany przez aplikację (**Theme**). Po naciśnięciu przycisku **OK** zostaje utworzony nowy, pusty projekt o podanej nazwie.



Rys. 8. Fragment zakładki Projekty w narzędziu MIT App Inventor

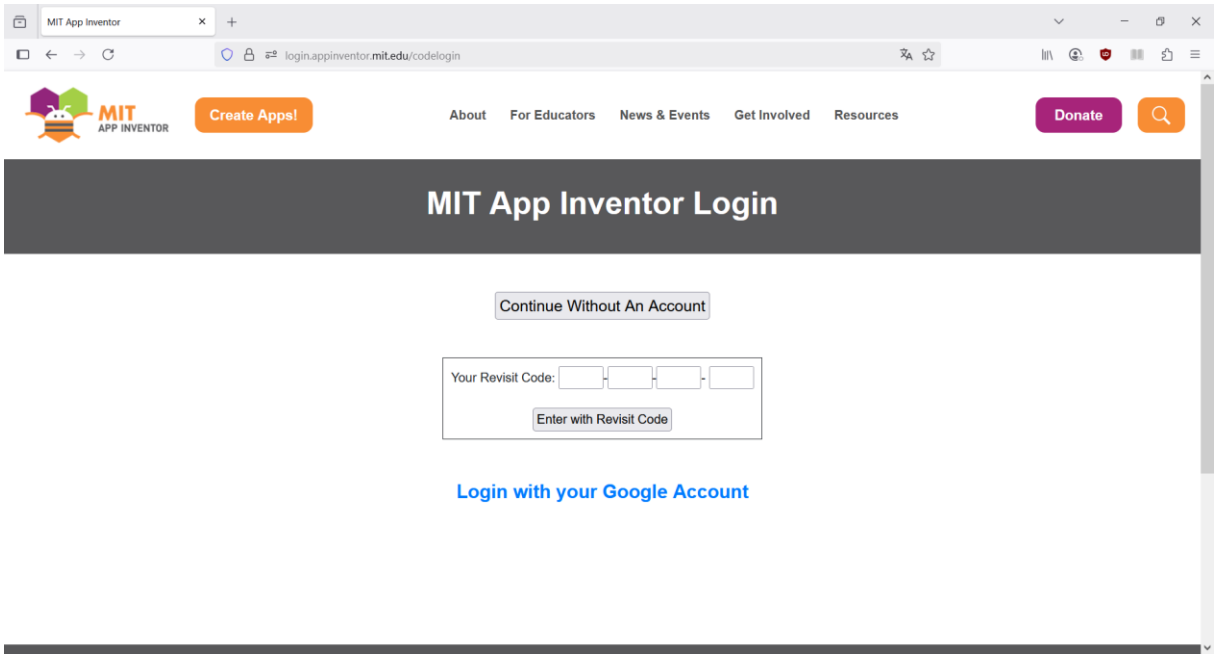
[Tekst alternatywny. Rysunek 8 przedstawia screen ekranu, na którym znajduje się górny fragment zakładki Projekty w narzędziu MIT App Inventor. Język został zmieniony na polski. Brak jest jakichkolwiek projektów.]



Rys. 9. Tworzenie nowego projektu w narzędziu MIT App Inventor

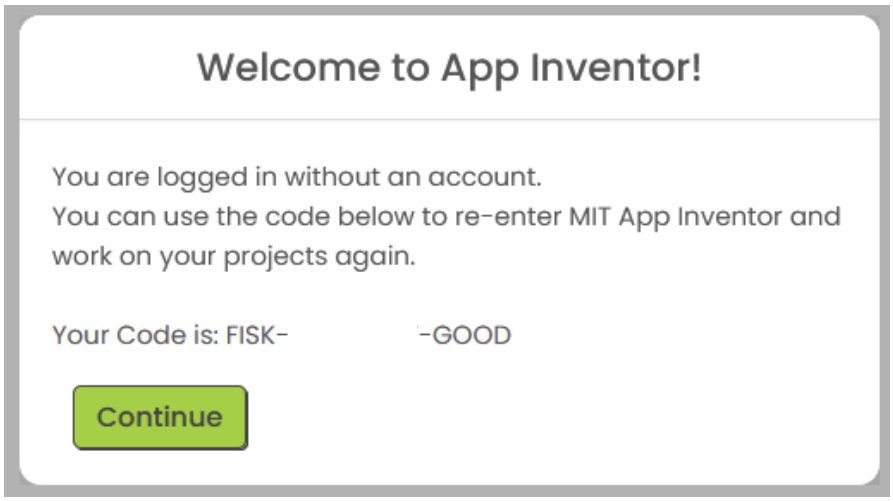
[Tekst alternatywny. Rysunek 9 przedstawia okno tworzenia nowego projektu w narzędziu MIT App Inventor. Można tu podać jego nazwę, wybrać zestaw potrzebnych komponentów i motyw.]

Alternatywnie, istnieje możliwość pracy w narzędziu MIT App Inventor bez posiadania konta Google. W tym celu należy skorzystać ze strony internetowej ze specjalnie przygotowaną wersją platformy: https://code.appinventor.mit.edu (rys. 10). Po wybraniu opcji **Continue Without An Account** użytkownik otrzymuje unikalny kod (rys. 11), który należy zachować. Podając go na wspomnianej przed chwilą stronie internetowej w miejscu **Your Revisit Code** (por. rys. 10), gdzie po naciśnięciu przycisku **Enter with Revisit Code** możliwa jest dalsza praca z utworzonymi projektami; nie są one jednak przypisane do konkretnego konta Google. Utrata kodu oznacza również utratę dostępu do tych projektów. Wersja ta może mieć jednak ograniczone funkcje.



Rys. 10. Strona internetowa https://code.appinventor.mit.edu

[Tekst alternatywny. Rysunek 10 przedstawia screen ekranu strony internetowej o adresie https://code.appinventor.mit.edu otwartej w przeglądarce internetowej Mozilla Firefox, gdzie można zalogować się do narzędzia MIT App Inventor bez posiadania konta Google, tylko wykorzystując unikalny kod.]

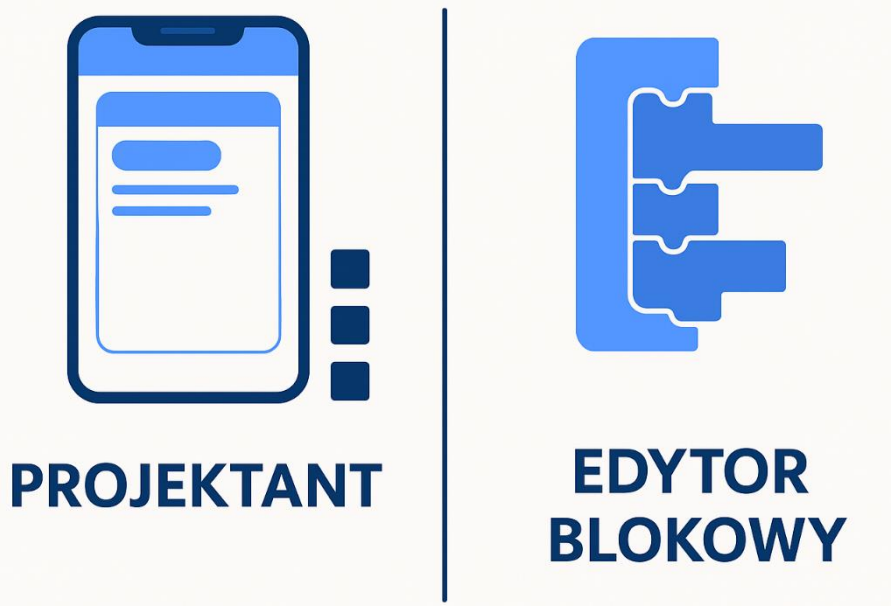


Rys. 11. Logowanie bez konta Google przy wykorzystaniu unikalnego kodu

[Tekst alternatywny. Rysunek 11 przedstawia fragment okna narzędzia MIT App Inventor, gdzie widoczny jest unikalny kod pozwalający na zalogowanie się do tej platformy bez posiadania konta Google.]

2.3. Omówienie środowiska pracy

Po utworzeniu nowego projektu lub otwarciu już istniejącego, można przystąpić do pracy nad budową aplikacji mobilnej. W środowisku MIT App Inventor proces jej tworzenia odbywa się w dwóch oddzielnych przestrzeniach roboczych – rys. 12. Pierwsza z nich to tryb **Projektant**, w którym definiuje się wygląd aplikacji – w tym celu rozmieszcza się wybrane komponenty interfejsu użytkownika na „wirtualnym urządzeniu" oraz ustawia się ich właściwości. Druga przestrzeń to **Edytor Blokowy**, w którym definiuje się (czyli „koduje") logikę działanie programu – w tym celu łączy się ze sobą odpowiednie bloki niczym puzzle. Warstwa wizualna jest więc wyraźnie oddzielona od logiki działania aplikacji.

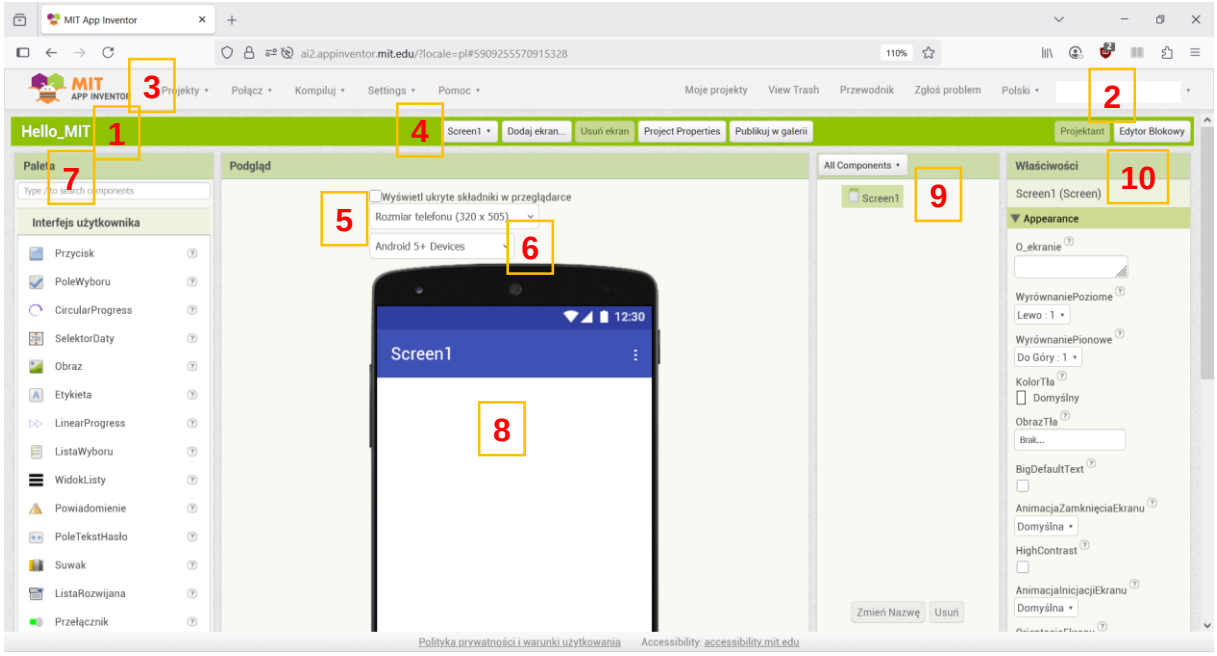


Rys. 12. Dostępne tryby pracy na platformie MIT APP Inventor (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

[Tekst alternatywny. Rysunek 12 przedstawia dwa podstawowe tryby pracy w narzędziu MIT App Inventor: po lewej stronie – Projektant (gdzie tworzy się wygląd interfejsu użytkownika; ikony reprezentują komponenty i wirtualne urządzenie), a po prawej – Edytor Blokowy (w którym za pomocą bloków „programuje się" logikę działania aplikacji).]

2.3.1. Tryb Projektant

Na rys. 13 przedstawiono interfejs użytkownika środowiska MIT App Inventor w trybie **Projektant** po utworzeniu nowego projektu (por. rys. 9) o nazwie Hello_MIT. Jak zaznaczono powyżej, to właśnie tutaj tworzy się pożądany wygląd aplikacji mobilnej.



Rys. 13. Narzędzie MIT APP Inventor – Tryb Projektant

[Tekst alternatywny. Rysunek 13 obejmuje screen ekranu przedstawiający narzędzie MIT App Inventor w trybie Projektant – tu projekt nazywa się Hello_MIT. W centralnej części znajduje się „wirtualny telefon, na którym można zobaczyć wygląd układu aplikacji. Po lewej stronie widoczna jest Paleta z kategoriami komponentów (tu rozwinięta jest paleta o nazwie Interfejs użytkownika, gdzie znajduje się m.in. Przycisk). Po prawej stronie telefonu najpierw widoczny jest Panel komponenty (zawierający jeden ekran **Screen1**), a następnie – panel Właściwości, w którym można definiować parametry zaznaczonego elementu. Na górze interfejsu znajdują się są przyciski nawigacyjne, np. Projekty, Połącz.]

W trybie **Projektant** znajdują się (por. rys. 13):

1. Nazwa projektu – tu Hello_MIT.
2. Przełączanie pomiędzy trybami **Projektant** oraz **Edytor Blokowy**.
3. W górnej części – swoiste menu nawigacyjne z grupami poleceń:
    a. **Projekty** – pozwala na zarządzanie projektem:

i. **Moje projekty** – umożliwia powrót do zakładki **Projekty**.
ii. **Rozpocznij nowy projekt** – tworzenie nowego projektu od podstaw.
iii. **Importuj projekt (.aia) z mojego komputera …** – import pliku ***.aia** (zawierającego kompletny zapis projektu aplikacji – wygląd i bloki kodu, umożliwiający jego przenoszenie i udostępnianie) z dysku.
iv. **Importuj projekt (.aia) z repozytorium …** – import pliku ***.aia** z repozytorium.
v. **Usuń projekt** – usunięcie bieżącego projektu.
vi. **Zapisz projekt** – zapis projektu.
vii. **Zapisz projekt jako …** – zapis kopii projektu pod podaną nazwą; praca nad projektem kontynuowana jest w tej kopii.
viii. **Punkt kontrolny** – zapis kopii projektu pod podaną nazwą; praca nad projektem kontynuowana jest w oryginalnym projekcie, a nie w tej kopii.
ix. **Project Properties**… – właściwości projektu; ustawienie podstawowych informacji i parametrów dotyczących całej aplikacji mobilnej (rys. 14), m.in. nazwa aplikacji (która jest widoczna zarówno na liście projektów, jak i po instalacji aplikacji na urządzeniu), możliwość ustawienia ikony aplikacji, rozmiar aplikacji (np. responsywny), motyw i kolory używane przez program, czy też kod i nazwa wersji aplikacji (istotne w kontekście udostępnienia aplikacji w Sklepie Play).
x. **Eksportuj wybrany projekt (.aia) do mojego komputera** – zapis kompletnego projektu w pliku .aia (kopia zapasowa) na komputerze.

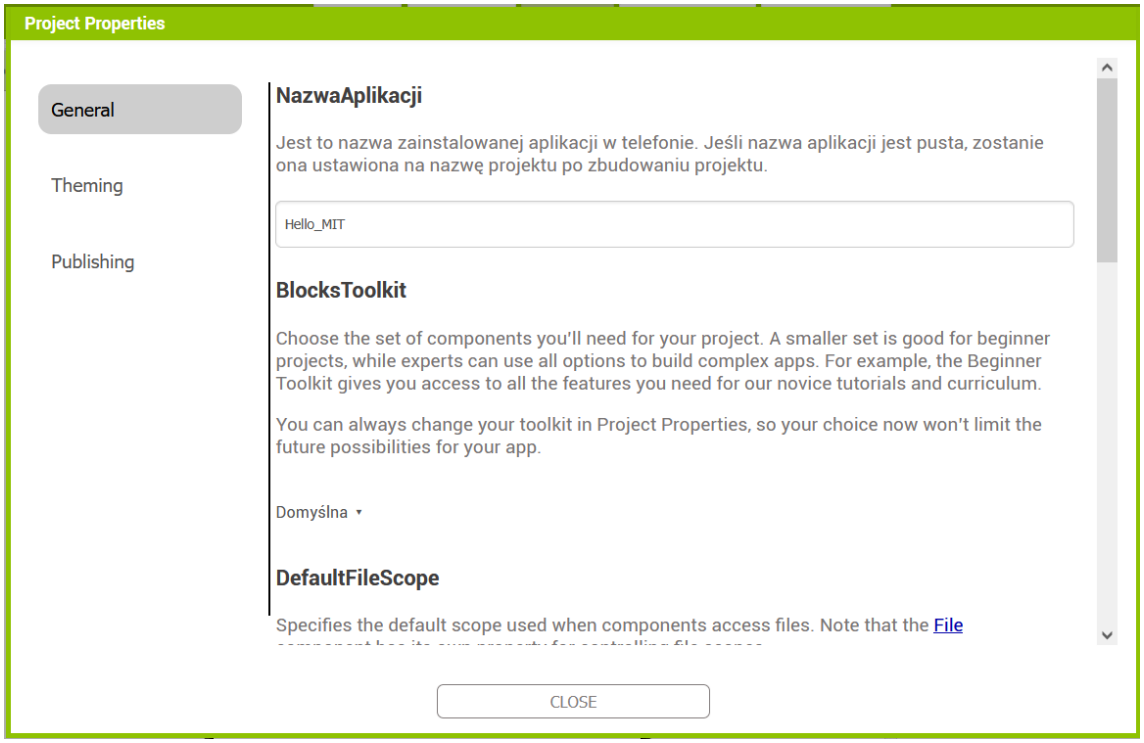
b. **Połącz** – połączenia środowiska MIT App Inventor z urządzeniem/ emulatorem w celu przetestowania aplikacji:
i. **AI Companion (WiFi)** – pozwala na testowanie aplikacji bezpośrednio na urządzeniu mobilnym w czasie rzeczywistym z zainstalowaną aplikacją MIT AI2 Companion; użytkownik musi zeskanować kod QR lub ręcznie przepisać kod.
ii. **Emulator** – pozwala na testowanie aplikacji na emulatorze telefonu z systemem Android zainstalowanym na komputerze. Opcja ta nie wymaga fizycznego urządzenia mobilnego.
iii. **USB** – umożliwia testowanie aplikacji w czasie rzeczywistym na fizycznym urządzeniu połączonym z komputerem za pomocą kabla USB.

iv. **Refresh Companion Screen** – odświeża ekran w aplikacji MIT AI2 Companion bez konieczności ponownego połączenia, co może być przydatne po wprowadzeniu zmian.
v. **Zresetuj połączenie** – kończy (zamyka) aktualne połączenie z urządzeniem mobilnym lub emulatorem.
vi. **Twardy reset** – wymusza całkowite rozłączenie oraz zresetowanie sesji połączeniowej, co może być przydatne, gdy wystąpią błędy połączenia.

c. **Kompiluj** – wygenerowanie finalnej wersji aplikacji na urządzenie mobilne, np. w formacie **.apk** na system Android; taką aplikację można zainstalować na fizycznym urządzeniu.
d. **Settings** (Ustawienia) – zmiana ustawień konfiguracyjnych.
e. **Pomoc** – oprócz wyświetlenia pomocy, można też tu m.in. dokonać aktualizacji aplikacji MIT AI2 Companion do najnowszej wersji.
f. **Moje projekty** – powrót do listy wszystkich projektów Użytkownika.
g. **View Trash** – przejście do „kosza”, gdzie znajdują się usunięte projekty.
h. **Przewodnik** – przejście do dokumentacji narzędzia MIT App Inventor.
i. **Zgłoś problem** – przejście do forum narzędzia MIT App Inventor.
j. **Wybór języka**.
k. **Informacje o koncie Google** – nazwa, możliwość wylogowania bądź też usunięcia konta.

4. **Ekrany** – możliwość dodawania, usuwania i przełączania się pomiędzy ekranami aplikacji; obok jest przycisk **Project Properties**.
5. Zmiana rozmiaru ekranu urządzenia widocznego w **Projektancie**, co pozwala na symulowanie różnych urządzeń. Do wyboru są trzy opcje: **Phone** (telefon – 320 x 505), **Tablet** (480 x 675) i **monitor** (768 x 1024). Dzięki temu można sprawdzić, jak aplikacja będzie wyglądać na różnych urządzeniach. Należy pamiętać, że urządzenia mobilne różnią się między sobą nie tylko rozmiarami ekranów – telefony komórkowe, tablety, telewizory itp. – ale także ich parametrami, np. gęstością pikseli (dpi). Wymiary są tutaj wyrażane w jednostkach niezależnych od tej gęstości – dp (ang. *density independent pixel*).
6. Wybór wersji mobilnego systemu operacyjnego, na której aplikacja będzie testowania wizualnie (np. różne wersje Androida lub iOS).
7. **Paleta komponentów** – zbiór komponentów służących do budowy interfejsu aplikacji. Są one pogrupowane w tematyczne kategorie (np. **Interfejs użytkownika**, **Układ**, **Media** – por. w dalszej części materiałów), a ich dodawanie odbywa się metodą „przeciągnij i upuść”.

8. **Wirtualny ekran urządzenia** – pełni rolę podglądu projektu; na nim są rozmieszczane wybrane komponenty interfejsu użytkownika. Dzięki temu można na bieżąco obserwować, jak aplikacja będzie wyglądać na ekranie urządzenia mobilnego i dostosowywać jej układ oraz wygląd.
9. **Komponenty** – hierarchiczna lista wszystkich komponentów, z których zbudowany jest dany ekran aplikacji. Można tutaj zmieniać nazwy tych komponentów bądź je usuwać, a także wybierać je w celu dalszej konfiguracji.
10. **Właściwości** – panel pozwalający na definiowanie parametrów zaznaczonego komponentu. Zawartość okna zmienia się kontekstowo w zależności od typu komponentu.
11. **Media** – nie jest widoczny na rys. 13; panel, w którym przechowywane są wszystkie pliki multimedialne wykorzystywane w aplikacji (np. obrazy i dźwięki).

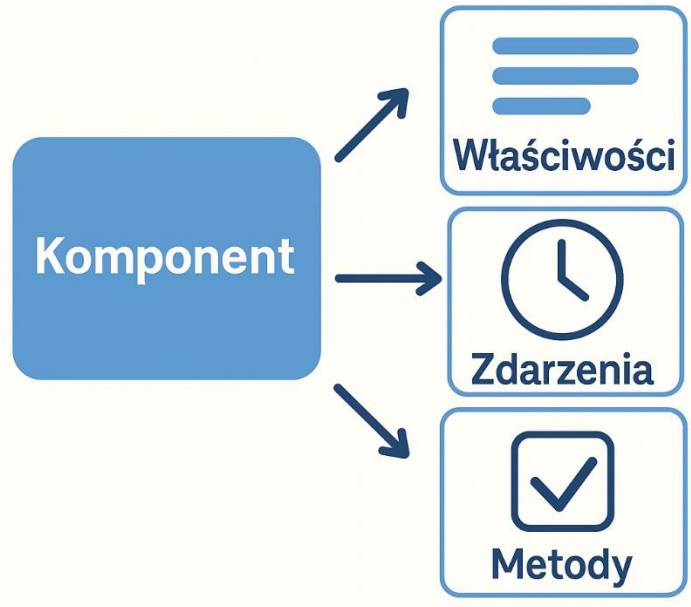


Rys. 14. Okno Project Properties – właściwości projektu

[Tekst alternatywny. Rysunek 14 przedstawia okno Project Properties w narzędziu MIT App Inventor. Pozwala ono na zdefiniowane właściwości projektu, tj. ustawienie podstawowych informacji i parametrów dotyczących całej aplikacji mobilnej, np. nazwa aplikacji. Po lewej stronie widoczne są trzy zakładki: General, Theming i Publishing.]

2.3.2. Komponenty – właściwości, zdarzenia i metody

Swoistym kluczem pozwalającym na efektywne budowanie aplikacji w środowisku MIT App Inventor jest zarówno znajomość dostępnych komponentów, jak też świadomość, że każdy z nich posiada swoje unikalne **właściwości**, **zdarzenia** i **metody** – rys. 15. Komponenty, w zależności od tego do czego służą, mają odmienne właściwości, zdarzenia i metody (ich dostępność jest więc kontekstowa), a niektóre z nich w ogóle mogą nie posiadać np. żadnych ustawień wizualnych lub nie udostępniać żadnych metod.



Rys. 15. Właściwości, zdarzenia i metody komponentów (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

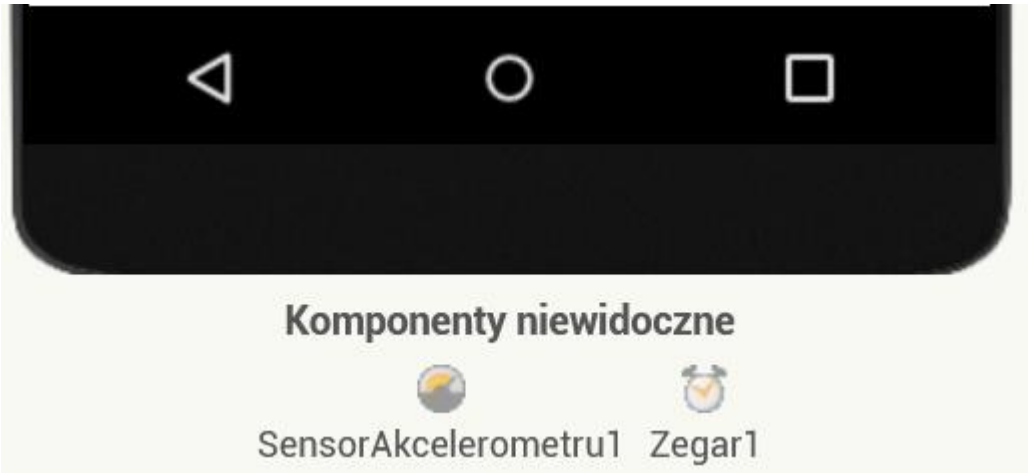
[Tekst alternatywny. Rysunek 15 przedstawia ideę komponentu w MIT App Inventor. Po lewej stronie, w niebieskim prostokącie z białym napisem, znajduje się słowo „Komponent". Strzałki prowadzą do trzech ramek po prawej stronie: „Właściwości" – symbolizuje cechy, które można ustawić (np. kolor, rozmiar), „Zdarzenia" – oznacza sytuacje, na które komponent może reagować (np. kliknięcie) oraz „Metody" – przedstawia działania, które można wywołać (np. ukrycie komponentu).]

Właściwości (ang. *properties*) to pewne cechy komponentu, które określają jego wygląd i zachowanie (np. kolor, rozmiar, czy tekst wyświetlany na ekranie). **Zdarzenia** (ang. *event*), to sytuacje, na które komponent może reagować (np. kliknięcie przycisku, czy zmiana położenia); programista może zaprojektować reakcję aplikacji na te zdarzenia. **Metody** (ang. *methods*) to swoiste akcje, które można

wywołać, aby zmienić stan komponentu lub wykonać określone zadanie, np. odtworzenie dźwięku.

Jak wspomniano już wcześniej, w narzędziu MIT App Inventor komponenty są pogrupowane w tematyczne grupy, które ułatwiają odnalezienie potrzebnych elementów podczas projektowania aplikacji. Niestety, przedstawienie wszystkich komponentów oraz ich właściwości, zdarzeń i metod wymagałoby kilkudziesięciu stron, co znacznie wykracza poza ramy niniejszego opracowania – poniżej poddano je krótkiej charakterystyce oraz przedstawiono wybrane komponenty. Każdorazowo podano link do dokumentacji technicznej, gdzie można odnaleźć wszelkie potrzebne informacje.

Część z komponentów w MIT App Inventor to komponenty niewidoczne (niewizualne), których nie widać bezpośrednio na ekranie urządzenia podczas działania programu, ale pełnią ważne funkcje operacyjne „w tle”. Przy domyślnych ustawieniach, można je odnaleźć pod wirtualnym ekranem urządzenia – rys. 16.



Rys. 16. Komponenty niewidoczne (niewizualne)

[Tekst alternatywny. Rysunek 16 przedstawia komponenty niewidoczne (niewizualne) w MIT App Inventor. Na górze widać dolny fragment wirtualnego ekranu urządzenia w obszarze Projektant, a poniżej SensorAkcelerometru1 oraz Zegar1.]

Można wyróżnić następujące grupy palet:

- **Interfejs użytkownika** (ang. *User Interface*) – zawiera komponenty służące do budowy graficznego interfejsu aplikacji, które to umożliwiają komunikację użytkownika z aplikacją i są podstawą każdej interaktywnej aplikacji. Wybrane

z nich przestawiono w tabeli 1. Więcej na: http://ai2.appinventor.mit.edu/reference/components/userinterface.html.

Tabela 1. Wybrane komponenty w narzędziu MIT App Inventor z grupy Interfejs użytkownika

Komponent	Opis
 Przycisk	**Przycisk** – wraz z możliwością wykrywania kliknięć. Wiele aspektów jego wyglądu można zmienić, a także to, czy można go kliknąć (*włączony*/ *wyłączony*).
 PoleWyboru	**Pole wyboru** (*checkbox*) – wywołuje zdarzenie, gdy użytkownik go kliknie (zaznaczy). Komponenty te mogą wykrywać dotknięcia użytkownika i w odpowiedzi zmieniać stan logiczny. Istnieje wiele właściwości wpływających na jego wygląd.
 SelektorDaty	**Selektor daty** – przycisk, który po kliknięciu uruchamia okno dialogowe umożliwiające użytkownikowi na wybranie daty.
 Obraz	**Obraz** – komponent do wyświetlania obrazów. Plik graficzny do wyświetlenia i inne aspekty wyglądu obrazu można określić w **Projektancie** lub w **Edytorze blokowym**.
 Etykieta	**Etykieta** – wyświetla fragment tekstu określony przez właściwość *Tekst*. Inne właściwości, z których wszystkie można ustawić w **Projektancie** lub **Edytorze blokowym**, kontrolują wygląd i rozmieszczenie tekstu.
 ListaWyboru	**Lista wyboru** – przycisk, który po kliknięciu wyświetla listę tekstów do wyboru przez użytkownika. Teksty można określić za pomocą **Projektanta** lub **Edytora blokowego**, ustawiając właściwość *ElementsFromString* (na przykład: wybór 1, wybór 2, wybór 3) lub ustawiając właściwość *Elementy* na *Listę* w *Edytorze Blokowym*.
 WidokListy	**Widok listy** – komponent wyświetlający listę elementów tekstowych. Listę można ustawić za pomocą właściwości *ElementsFromString* lub za pomocą bloku *Elementy* w *Edytorze Blokowym*.
 Powiadomienie	**Powiadomienie** – wyświetla okna dialogowe alertów, komunikaty i alerty tymczasowe oraz tworzy wpisy dziennika systemu Android za pomocą określonych metod. Komponent niewizualny.
 PoleTekstHasło	**PoleTekstHasło** – tworzy pole do wprowadzania haseł. Jest to to samo, co zwykły komponent **PoleTekstowe**, z wyjątkiem tego, że nie wyświetla znaków wpisanych przez użytkownika, tzn. ukrywa wpisany w nim tekst.
 Suwak	**Suwak** – komponent, który dodaje „przeciągany palcem” suwak. Można go dotknąć i przeciągnąć w lewo lub w prawo, aby ustawić pożądaną pozycję.
 ListaRozwijana	**Lista rozwijana** – komponent wyświetlający wyskakujące okienko z listą elementów. Elementy można ustawić w **Projektancie** lub **Edytorze Blokowym**, definiując właściwość *ElementsFromString*.

Komponent	Opis
 Przełącznik	**Przełącznik** – tworzy pole wyboru, które wywołuje zdarzenie, gdy użytkownik je kliknie. Istnieje wiele właściwości wpływających na jego wygląd, które można ustawić w **Projektancie** lub **Edytorze blokowym**.
 PoleTekstowe	**Pole tekstowe** – tworzy pole do wprowadzania tekstu przez użytkownika. Początkowa lub wprowadzona przez użytkownika wartość tekstu znajduje się we właściwości *Text*. Więcej na: http://ai2.appinventor.mit.edu/reference/components/userinterface.html#TextBox.
 SelektorCzasu	**Selektor czasu** – komponent, który po kliknięciu uruchamia wyskakujące okno dialogowe, co umożliwia wybór czasu.
 WebViewer	Komponent do przeglądania stron internetowych. Nie jest to „pełna przeglądarka", np. naciśnięcie klawisza *Wstecz* w telefonie spowoduje zamknięcie aplikacji zamiast cofnięcia się zgodnie z historią przeglądarki. Adres URL można określić w **Projektancie** lub w **Edytorze blokowym**. Widok można ustawić tak, aby śledził linki, gdy są one dotknięte, a użytkownicy mogą wypełniać formularze internetowe.

- **Układ** (ang. *Layout*) – obejmuje komponenty pozwalające na organizację i rozmieszczanie elementów interfejsu na ekranie, np. w pionie lub poziomie. Dzięki nim można tworzyć przejrzyste i responsywne układy graficzne. Wybrane z nich przestawiono w tabeli 2. Więcej na: http://ai2.appinventor.mit.edu/reference/components/layout.html.

Tabela 2. Wybrane komponenty w narzędziu MIT App Inventor z grupy Układ

Komponent	Opis
 UkładPoziomy	**Układ poziomy –** element grupujący, w którym umieszcza się komponenty, które powinny być wyświetlane od lewej do prawej.
 UkładPoziomyPrzewijalny	**Układ poziomy przewijalny** – element grupujący, w którym umieszcza się komponenty, które powinny być wyświetlane od lewej do prawej. Wersja przewijalna.
 UkładTabelaryczny	**Układ tabelaryczny** – element grupujący, w którym umieszcza się komponenty, które powinny być wyświetlane w formie tabelarycznej.
 UkładPionowy	**Układ pionowy** – element grupujący, w którym umieszcza się komponenty, które powinny być wyświetlane jeden pod drugim.
 UkładPionowyPrzewijany	**Układ pionowy przewijalny** – element grupujący, w którym umieszcza się komponenty, które powinny być wyświetlane jeden pod drugim. Wersja przewijalna.

- **Media** – obejmuje komponenty umożliwiają wzbogacenie aplikacji o funkcje multimedialne np. do obsługi dźwięku, obrazu, nagrywania audio i wideo, a także kamery. Aplikacja MIT App Inventor dla systemu Android ogranicza łączny rozmiar aplikacji (plik *.apk) do 5 MB, z czego nie wszystko jest dostępne dla plików multimedialnych (wideo, audio i dźwiękowe). Jeśli pliki multimedialne są zbyt duże, mogą pojawić się błędy podczas pakowania lub instalowania aplikacji. Wybrane z nich przestawiono w tabeli 3. Więcej na: http://ai2.appinventor.mit.edu/reference/components/media.html.

Tabela 3. Wybrane komponenty w narzędziu MIT App Inventor z grupy Media

Komponent	Opis
 KameraWideo	**Kamera wideo** – komponent do nagrywania wideo za pomocą kamery urządzenia. Po nagraniu wideo, nazwa pliku zawierającego film jest dostępna w telefonie jako argument zdarzenia *AfterRecording*.
 AparatFotograficzny	**Aparat fotograficzny** – komponent służący do robienia zdjęcia za pomocą aparatu urządzenia. Po zrobieniu zdjęcia, nazwa pliku zawierającego obraz jest dostępna w telefonie jako argument do zdarzenia *AfterPicture*.
FilePicker	**Selektor plików** – komponent typu przycisk, który po kliknięciu przez użytkownika wyświetla okno pozwalające na wybranie pliku z systemu mobilnego.
 SelektorObrazów	**Selektor obrazów** – tworzy specjalny przycisk. Gdy użytkownik go kliknie, pojawia się galeria obrazów urządzenia, a użytkownik może wybrać obraz. Po wybraniu obrazu jest on zapisywany, a właściwość *Selected* stanowi nazwę pliku, w którym przechowywany jest obraz. Aby nie zapełniać pamięci, zostanie zapisanych maksymalnie 10 zdjęć. Wybór większej liczby obrazów spowoduje usunięcie poprzednich obrazów, w kolejności od najstarszego do najnowszego.
 Gracz	**Odtwarzacz** – komponent multimedialny, który odtwarza dźwięk i kontroluje wibracje telefonu. Najlepszy w przypadku dłuższych plików. Nazwa obiektu multimedialnego jest określona we właściwości *Source*, którą można ustawić w **Projektancie** lub **Edytorze blokowym**. Czas trwania wibracji jest określony w edytorze bloków w milisekundach (tysięczne części sekundy).
 Dźwięk	**Dźwięk** – komponent multimedialny, który odtwarza pliki dźwiękowe i opcjonalnie wibruje przez liczbę milisekund określonych w **Edytorze bloków**. Nazwę pliku dźwiękowego do odtworzenia można określić w **Projektancie** lub w **Edytorze bloków**. Jest najlepszy dla krótkich plików dźwiękowych, takich jak jako efekty dźwiękowe, podczas gdy komponent **Gracz** jest bardziej wydajny dla dłuższych dźwięków, takich jak piosenki.
 RejestratorDźwięku	**Rejestrator dźwięku** – komponent multimedialny, który rejestruje dźwięk.

Komponent	Opis
 RozpoznawanieMowy	**Rozpoznawanie mowy** – komponent do używania funkcji rozpoznawania mowy. Służy do konwersji z mowy na tekst.
 TekstNaMowę	**Tekst na mowę** – komponent przekształcający dany tekst na mowę.
 Translator	**Translator** – komponent do tłumaczenia słów i zdań między różnymi językami. Wymaga dostępu do Internetu, ponieważ wykorzystuje serwer MIT. Należy określić język źródłowy i docelowy.
 OdtwarzaczWideo	**Odtwarzacz video** – komponent multimedialny umożliwiający odtwarzanie filmów. Po uruchomieniu aplikacji zostaje on wyświetlany jako prostokąt na ekranie. Jeśli użytkownik dotknie prostokąt, pojawią się kontrolki do odtwarzania/pauzy, przeskakiwania do przodu i przeskakiwania do tyłu w obrębie wideo. Pliki wideo powinny być w formatach 3GPP (.3gp) lub MPEG-4 (.mp4). Aplikacja MIT App Inventor dla systemu Android zezwala tylko na pliki wideo o rozmiarze poniżej 1 MB.

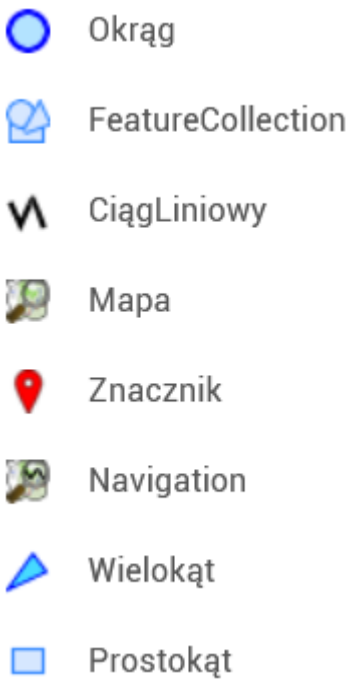
- **Rysowanie i animacja** (ang. *Drawing and Animation*) – zawiera komponenty umożliwiające tworzenie dynamicznych elementów graficznych oraz prostych interakcji wizualnych, m.in. rysowanie na ekranie, przemieszczanie obiektów, co pozwala także tworzyć animacje i gry. Wybrane z nich przestawiono w tabeli 4. Więcej na: http://ai2.appinventor.mit.edu/reference/components/animation.html.

Tabela 4. Wybrane komponenty w narzędziu MIT App Inventor z grupy Rysowanie i animacja

Komponent	Opis
 Piłka	**Piłka** – okrągły „duszek", który może być umieszczony na **Canvasie**, gdzie może reagować na dotyk i przeciąganie, interakcję z innymi „duszkami" (**Piłka** lub **Ikonka**) i krawędziami **płótna**; porusza się zgodnie z zadanymi parametrami.
 Canvas	**Canvas** (**płótno**) – dwuwymiarowy, dotykowy, prostokątny panel, na którym można rysować i przesuwać „duszki". Dowolną lokalizację na **Canvasie** można określić jako parę wartości (*X*, *Y*), gdzie: *X* to liczba pikseli od lewej krawędzi **płótna**, a *Y* to liczba pikseli od górnej krawędzi **płótna**. Istnieją zdarzenia informujące, kiedy i gdzie dotknięto **Canvas** lub „duszki" (**Piłka** lub **Ikonka**) i gdzie zostały przeciągnięte. Istnieją również metody rysowania punktów, linii i okręgów.

Komponent	Opis
Ikonka	**„Duszek”** – **ikonka**, którą można umieścić na **Canvasie**, gdzie może reagować na dotyk i przeciąganie, interakcję z innymi „duszkami” (**Piłka** lub **Ikonka**) i krawędzią **płótna**; porusza się zgodnie z zadanymi wartościami właściwości. Różnica między komponentami **Piłka**, a **Ikonka** polega na tym, że ta druga może uzyskać swój wygląd z pliku obrazu, podczas gdy wygląd **piłki** może zostać zmieniony tylko przez zmianę jej właściwości *Kolor* i *Promień*.

- **Mapy** (ang. *Maps*) – zawiera komponenty geolokalizacyjne, służące do wyświetlania i interakcji z mapami bezpośrednio w aplikacji – rys. 17. Umożliwia m.in. wyświetlanie pozycji użytkownika, dodawanie znaczników, śledzenie tras oraz prezentowanie danych przestrzennych, co czyni ją szczególnie przydatną w aplikacjach lokalizacyjnych i nawigacyjnych. Więcej na: https://ai2.appinventor.mit.edu/reference/components/maps.html.



Rys. 17. Komponenty z grupy Mapy

[Tekst alternatywny. Rysunek 17 przedstawia komponenty z grupy Mapy w narzędziu MIT App Inventor, np. Okrąg, FeatureCollection, Ciąg liniowy, Mapa.]

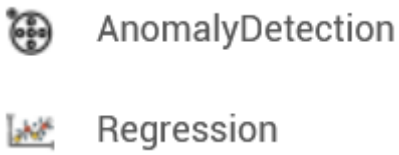
- **Charts** – paleta **Wykresy** umożliwia wizualizację danych w postaci wykresów, co może być szczególnie przydatne dla Studentów kierunku ***Inżynieria***

danych. Wybrane z nich przestawiono w tabeli 5. Więcej na: http://ai2.appinventor.mit.edu/reference/components/charts.html.

Tabela 5. Wybrane komponenty w narzędziu MIT App Inventor z grupy Charts

Komponent	Opis
Chart	**Wykres** – komponent pozwalający przedstawić dane w postaci jednego z 5 typów wykresów: liniowy, warstwowy, punktowy, słupkowy oraz kołowy, które można zmienić za pomocą właściwości *Rodzaj*. Ma wiele właściwości, które pozwalają zmienić wygląd wykresu.
ChartData2D	Komponent służący do przechowywania i zarządzania danymi dwuwymiarowymi (*X*, *Y*), które są wyświetlane na **Wykresach**.
Trendline	Linia trendu – komponent wykorzystywany do dodawania linii trendu na **Wykresach**.

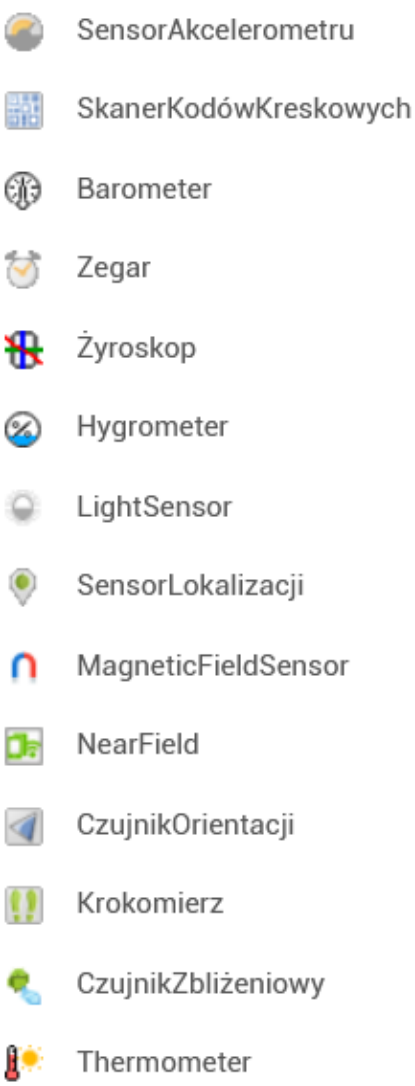
- **Data Science** – zawiera komponenty przydatne z punktu widzenia analiz data science – rys. 18. Na obecną chwilę umożliwiają one wykrywanie anomalii w danych oraz stosowanie modeli regresji. Więcej na: https://ai2.appinventor.mit.edu/reference/components/datascience.html.



Rys. 18. Komponenty z grupy Data Science

[Tekst alternatywny. Rysunek 18 przedstawia komponenty z grupy Data Science w narzędziu MIT App Inventor: AnomalyDetection i Regression.]

- **Czujniki** (ang. *Sensors*) – zawiera komponenty pozwalające na korzystanie z fizycznych czujników urządzenia mobilnego – rys. 19. Wybrane trzy z nich omówiono w tabeli 6. Więcej na: http://ai2.appinventor.mit.edu/reference/components/sensors.html.



Rys. 19. Komponenty z grupy Czujniki

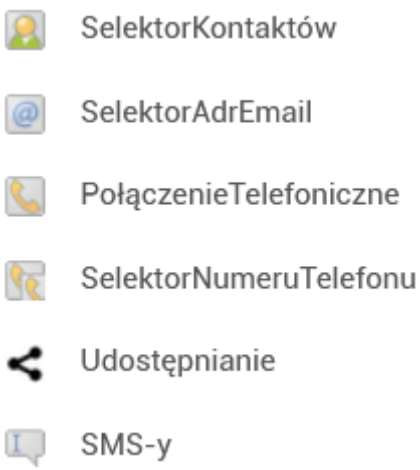
[Tekst alternatywny. Rysunek 19 przedstawia komponenty z grupy Czujniki w narzędziu MIT App Inventor, np. SensorAkcelerometru oraz SkanerKodówKreskowych.]

Tabela 6. Wybrane komponenty w narzędziu MIT App Inventor z grupy Czujniki

Komponent	Opis
Zegar	**Zegar** – komponent niewizualny, który zapewnia dostęp do czasu, mierzonego za pomocą wewnętrznego zegara telefonu. Może także uruchamiać stoper w regularnych odstępach czasu.
SkanerKodówKreskowych	**Skaner kodów QR** – komponent do używania skanera kodów kreskowych do odczytu kodu kreskowego; w wyniku zwracany jest ciąg znaków.

Komponent	Opis
SensorAkcelerometru	**Akcelerometr** – komponent niewizualny, który może wykrywać wstrząsy i mierzyć przyspieszenie (w przybliżeniu) w trzech wymiarach, przy użyciu jednostek SI (m/s^2). Składowe to: • *xAccel*: 0, gdy telefon spoczywa na płaskiej powierzchni; dodatni, gdy telefon jest pochylony w prawo (tzn. jego lewa strona jest uniesiona); ujemny, gdy telefon jest przechylony w lewo (tzn. jego prawa strona jest uniesiona), • *yAccel*: 0, gdy telefon spoczywa na płaskiej powierzchni; dodatni, gdy jego dolna część jest uniesiona; ujemny, gdy jego górna część jest uniesiona, • *zAccel* : równy -9.8 (grawitacja ziemi w metrach na sekundę do kwadratu), gdy urządzenie jest ustawione równolegle do ziemi z wyświetlaczem skierowanym do góry; 0, gdy prostopadle do ziemi; +9.8, gdy jest skierowane do dołu.

- **Społeczny** (ang. *Social*) – obejmuje komponenty umożliwiające interakcję z funkcjami urządzenia takimi jak: SMS, e-mail, telefon, kontakty, czy też udostępnianie treści – rys. 20. Więcej na: http://ai2.appinventor.mit.edu/reference/components/social.html.



Rys. 20. Komponenty z grupy Społeczny

[Tekst alternatywny. Rysunek 20 przedstawia komponenty z grupy Społeczny w narzędziu MIT App Inventor, np. SelektorKontaktów oraz SelektorAdrEmail.]

- **Przechowywanie** (ang. *Storage*) – obejmuje komponenty służące do zapisywania i odczytywania danych lokalnie lub w chmurze. Domyślnie aplikacje utworzone w MIT App Inventorze są inicjowane przy każdym uruchomieniu; jeśli aplikacja ustawia wartość jakiejś zmiennej, a użytkownik

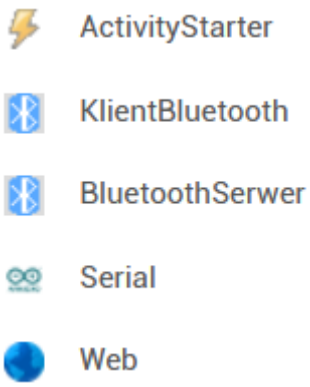
zamyka aplikację, wartość tej zmiennej nie zostaje zapamiętana przy następnym uruchomieniu aplikacji. Czasami istnieje potrzeba zapisu pewnych danych w pamięci trwałej urządzenia. Do dyspozycji jest kilka różnych komponentów. Wybrane z nich przedstawiono w tabeli 7. Więcej na: http://ai2.appinventor.mit.edu/reference/components/storage.html.

Tabela 7. Wybrane komponenty w narzędziu MIT App Inventor z grupy Przechowywanie

Komponent	Opis
 CloudDB	**CloudDB** – niewizualny komponent umożliwiający przechowywanie danych na serwerze baz danych, wykorzystując połączenie z Internetem (przy użyciu oprogramowania *Redis* – domyślnie dane są przechowywane na serwerze obsługiwanym przez MIT). Dzięki temu użytkownicy aplikacji mogą współdzielić dane.
 DataFile	**DataFile** – komponent niewizualny, pozwalający na odczyt danych w formatach CSV i JSON.
Plik	**Plik** – niewizualny komponent służący do zapisu i odczytu plików na urządzeniu. Dokładna lokalizacja, w której umieszczane są pliki zewnętrzne, jest wartością właściwości *Scope* – niezależnie od tego, czy aplikacja działa w programie MIT AI2 Companion, czy jest skompilowana.
 TinyDB	**TinyDB** – niewizualny, bardzo ważny komponent bazodanowy, pozwalający na trwałe przechowywanie określonych danych aplikacji w pamięci urządzenia. **TinyDB** to trwały magazyn danych dla aplikacji – tzn. przechowywane dane będą dostępne za każdym razem, gdy aplikacja jest uruchamiana. Przykładem może być gra, która zapisuje najwyższy wynik i pobiera go za każdym razem, gdy jest uruchamiana. Elementy danych (**wartości**) to łańcuchy przechowywane pod ***tagami***. Aby zapisać element danych, należy zdefiniować ***tag***, w którym ma być przechowywany. Na aplikację przypada tylko jeden magazyn danych – nawet, gdy jest wiele komponentów **TinyDB**, będą one korzystać z tego samego magazynu danych. Aby uzyskać efekt oddzielnych „przestrzeni do magazynowania", można użyć różnych kluczy (*Namespace*). Ponieważ każda aplikacja ma swój własny magazyn danych, więc nie można używać **TinyDB** do przekazywania danych między różnymi aplikacjami w telefonie. Można jednakże wykorzystywać go do przekazywania danych pomiędzy różnymi ekranami aplikacji wieloekranowej. Podczas testowania aplikacji, korzystając z emulatora lub oprogramowania MIT AI2 Companion, wszystkie aplikacje będą miały do dyspozycji tę samą bazę danych. Należy więc usuwać zawartość **TinyDB** za każdym razem, gdy zaczyna się pracę nad nową aplikacją.

Komponent	Opis
TinyWebDB	TinyWebDB – niewizualny komponent komunikujący się z usługą sieciową w celu przechowywania i pobierania informacji. Towarzysząca mu usługa sieciowa znajduje się pod adresem http://tinywebdb.appinventor.mit.edu. Komponent ten posiada metody do przechowywania wartości pod ***tagiem*** i pobierania ***wartości*** powiązanej z danym ***tagiem***.
Spreadsheet	**Arkusz kalkulacyjny** – niewizualny komponent do przechowywania i odbierania danych z arkusza kalkulacyjnego przy pomocy interfejsu API Arkuszy Google.

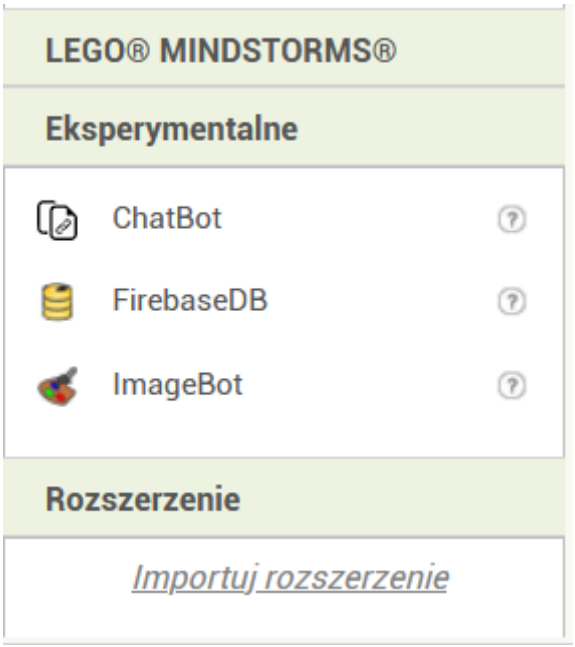
- **Łączność** (ang. *Connectivity*) – zawiera komponenty umożliwiające łączenie się z innymi urządzeniami lub usługami – rys. 21, np. przez Bluetooth, Internet (HTTP), Web API lub Wi-Fi. Pozwala więc na tworzenie aplikacji komunikujących się z zewnętrznymi systemami. Więcej na: http://ai2.appinventor.mit.edu/reference/components/connectivity.html.



Rys. 21. Komponenty z grupy Łączność

[Tekst alternatywny. Rysunek 21 przedstawia komponenty z grupy Łączność w narzędziu MIT App Inventor, np. ActivityStarter oraz KlientBluetooth].

- Pozostałe palety komponentów: **LEGO® MINDSTORMS®** (przeznaczona do sterowania robotami LEGO MINDSTORMS za pomocą aplikacji; umożliwia wysyłanie poleceń i odbieranie danych z czujników robota; https://ai2.appinventor.mit.edu/reference/components/legomindstorms.html), **Eksperymentalne** (https://ai2.appinventor.mit.edu/reference/components/experimental.html) i **Rozszerzenia** (umożliwia dodawanie własnych komponentów rozszerzających funkcjonalność MIT App Inventora; dzięki nim możliwe jest korzystanie z dodatkowych bibliotek i niestandardowych funkcji) – rys. 22.



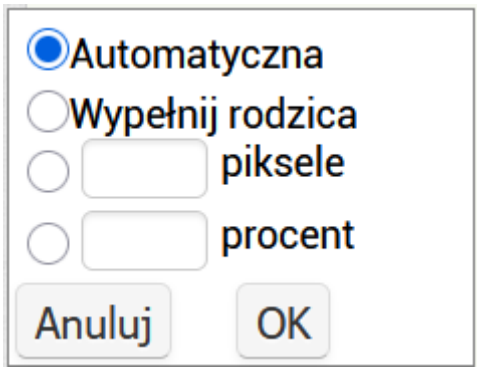
Rys. 22. Pozostałe grupy komponentów

[Tekst alternatywny. Rysunek 22 przedstawia pozostałe grupy komponentów w narzędziu MIT App Inventor: LEGO® MINDSTORMS®, Eksperymentalne oraz Rozszerzenie.]

Na koniec należy dodać, że dla większości komponentów **szerokość** (ang. *width*) oraz **wysokość** (ang. *height*) można zdefiniować jako (rys. 23):

- **Automatyczna** – narzędzie samodzielnie (automatycznie) dobiera właściwą wartość szerokości lub wysokości.
- **Wypełnij rodzica** (ang. *Fill parent*) – komponent przejmuje wielkość komponentu nadrzędnego. Rozmiar jest dobierany tak, aby wypełnić całą dostępną przestrzeń.
- **... piksele** – komponent posiada szerokość lub wysokość o podanej liczbie pikseli.
- **... procent** – komponent posiada szerokość lub wysokość zdefiniowaną w procentach (np. 50% szerokości ekranu).

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki” nr FERS.01.05-IP.08-0234/23

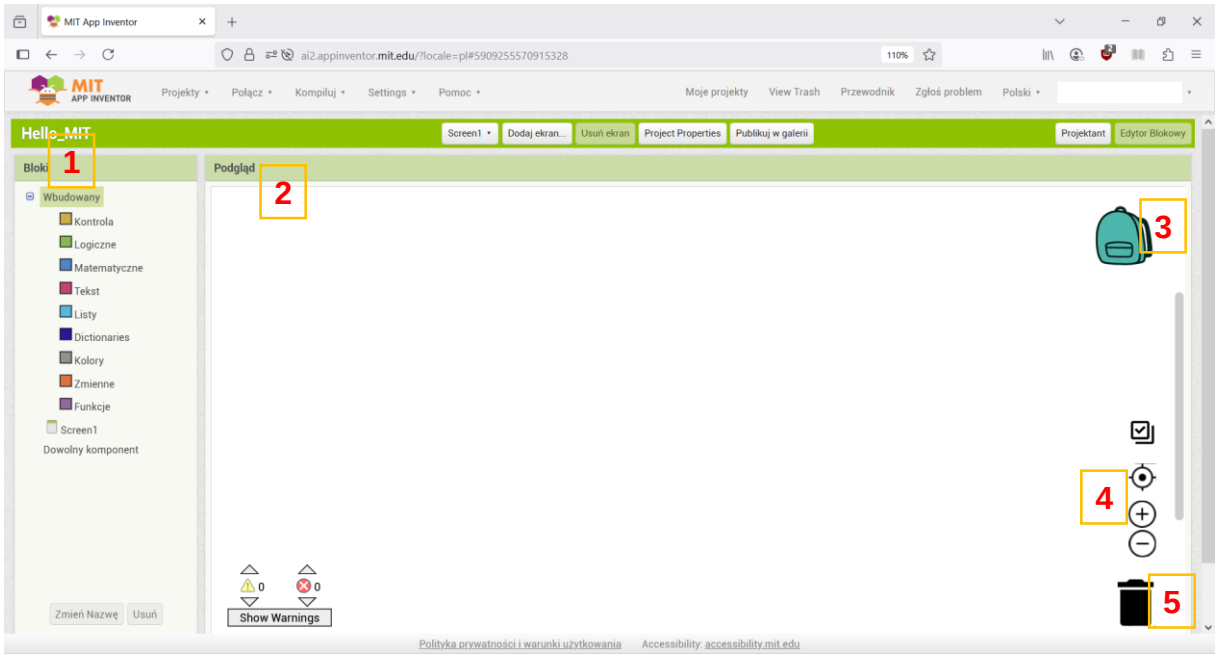


Rys. 23. Definiowanie wysokości i szerokości komponentów

[Tekst alternatywny. Rysunek 23 przedstawia definiowanie wysokości i szerokości komponentów w narzędziu MIT App Inventor.]

2.3.3. Tryb Edytor Blokowy

Na rys. 24 przedstawiono interfejs użytkownika środowiska MIT App Inventor w trybie **Edytor Blokowy** – projekt o nazwie Hello_MIT.



Rys. 24. Narzędzie MIT APP Inventor – Tryb Edytor Blokowy

[Tekst alternatywny. Rysunek 24 obejmuje screen ekranu przedstawiający narzędzie MIT App Inventor w trybie Edytor Blokowy – tu projekt nazywa się Hello_MIT.
W centralnej części znajduje się przestrzeń robocza (brak bloków kodu), a po lewej

stronie panel z grupami bloków do wykorzystania w projekcie: bloki wbudowane (Kontrola, Logiczne, Matematyczne, Tekst, Listy, Dictionaries, Kolory, Zmienne, Funkcje) i bloki komponentowe (Screen1).]

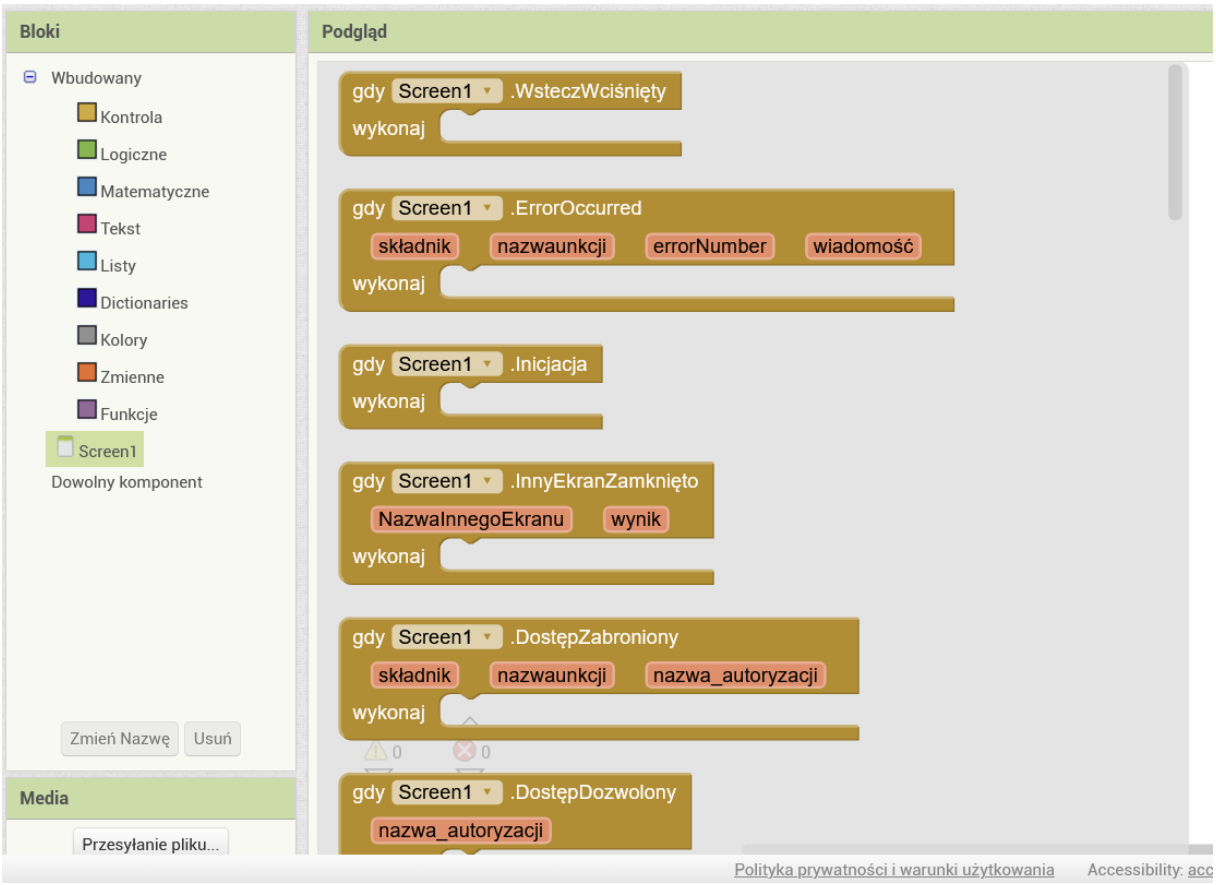
Jak wspomniano powyżej, to właśnie tutaj tworzy się logikę działania programu, poprzez łączenie ze sobą odpowiednich bloków kodu. Zaletę stanowi fakt, iż bloki pasują do siebie tylko wtedy, gdy są logicznie i składniowo poprawne, co zapobiega wielu potencjalnym błędom. Określa się przez to, jak aplikacja ma reagować na działania użytkownika, jakie operacje ma wykonywać oraz w jaki sposób przetwarzać dane. Można tu wyróżnić (por. rys. 24):

1. Paleta **Bloki** – obejmuje posortowane we właściwe kategorie bloki programistyczne, które dodaje się do obszaru roboczego metodą „przeciągnij i upuść”; bloki te są podzielone na główne grupy:
    a. **Bloki wbudowane** (ang. *Built-in blocks*) – są niezależne od konkretnego projektu i dostępne bez względu na to, jakie komponenty się w nim znajdują. Obejmują one podstawowe struktury programistyczne, takie jak: operatory logiczne i matematyczne, zmienne, instrukcje warunkowe, pętle, funkcje (procedury) oraz bloki do obsługi tekstów, list, czy kolorów. Po wybraniu odpowiedniej kategorii, po prawej stronie pojawiają się bloki, które do niej należą – np. na rys. 25 dla kategorii **Matematyczne**.

[Tekst alternatywny. Rysunek 25 obejmuje screen ekranu przedstawiający fragment narzędzia MIT App Inventor w trybie Edytor Blokowy z widocznymi blokami z grupy „Matematyczne”.]

Rys. 25. Przykładowe bloki z grupy „Matematyczne”

b. **Bloki komponentowe** – są generowane automatycznie dla każdego komponentu dodanego do projektu w trybie **Projektant**. Każdy komponent posiada swój własny zestaw bloków specyficznych dla jego zdarzeń, metod i właściwości, który jest dostosowany do jego funkcji i możliwości – np. na rys. 26 dla komponentu **Screen1**.



Rys. 26. Przykładowe bloki dla komponentu Screen1

[Tekst alternatywny. Rysunek 26 obejmuje screen ekranu przedstawiający fragment narzędzia MIT App Inventor w trybie Edytor Blokowy z widocznymi blokami dla komponentu Screen1.]

c. **Dowolny komponent** (ang. *Any component*) – obejmuje bloki, które umożliwiają tworzenie uniwersalnych procedur i operacji działających jednocześnie na wszystkich komponentach tego samego typu (m.in. dla dowolnego przycisku, pola tekstowego, etykiety itd.). Oznacza to, iż zamiast odnosić się do konkretnego komponentu, np. przycisku o nazwie **Przycisk1**, można zaprogramować operacje dla dowolnego przycisku, co pozwala na budowę bardziej elastycznych bloków kodu. Na rys. 27 przedstawiono przykładowe bloki dla komponentu **JakikolwiekPrzycisk**.

Rys. 27. Przykładowe bloki dla komponentu JakikolwiekPrzycisk

[Tekst alternatywny. Rysunek 27 obejmuje screen ekranu przedstawiający fragment narzędzia MIT App Inventor w trybie Edytor Blokowy z widocznymi przykładowymi blokami dla komponentu JakikolwiekPrzycisk.]

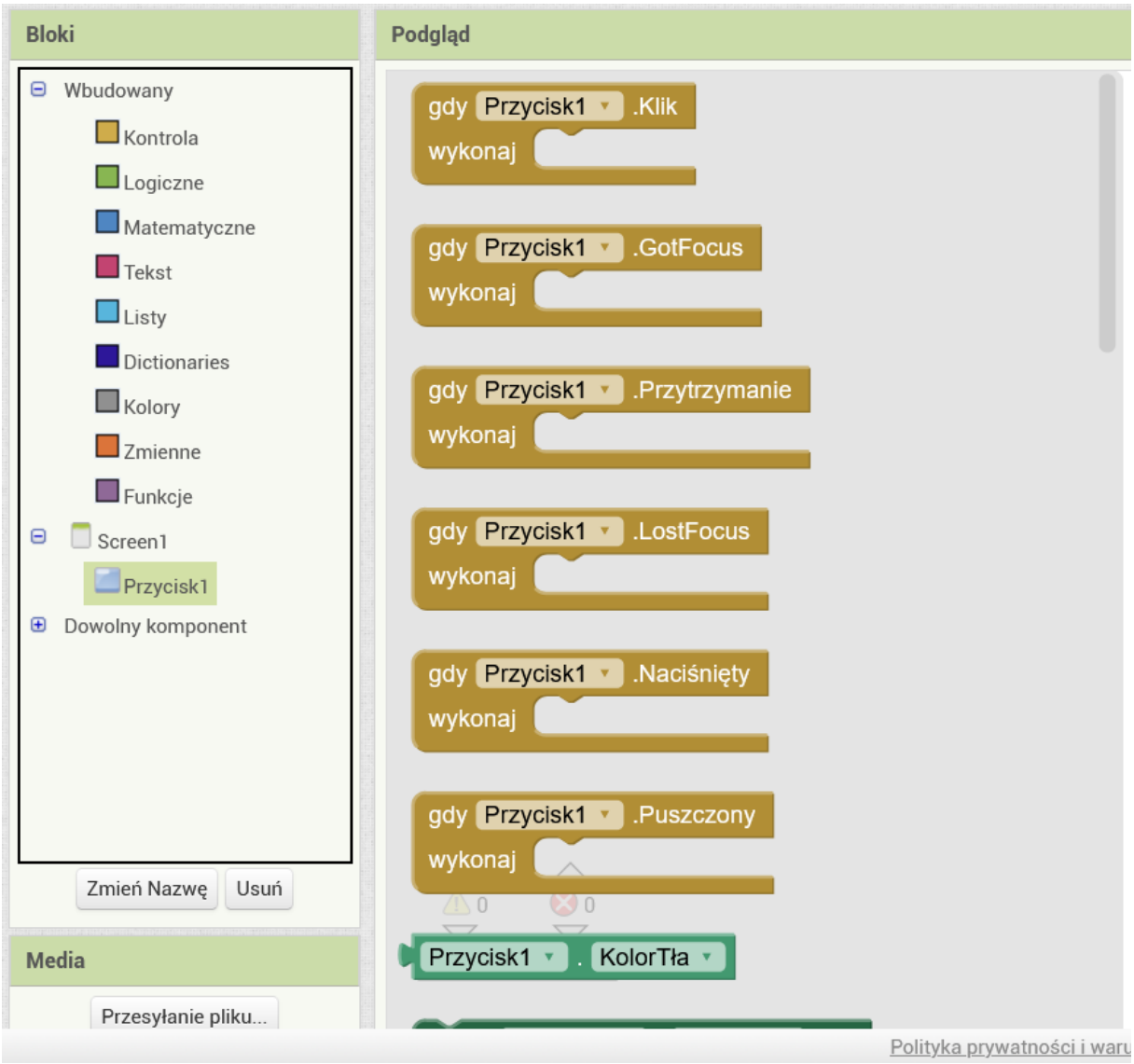
2. **Podgląd** (ang. *Viewer*) – obejmuje obszar roboczy, w którym umieszcza się i łączy ze sobą bloki kodu. W tym celu przeciąga się bloki z panelu po lewej stronie i składa je w struktury odpowiadające instrukcjom programistycznym.
3. **Plecak** – pozwala na wielokrotne wykorzystywanie tych samych fragmentów

bloków. Umożliwia kopiowanie bloków z jednego ekranu lub projektu, a następnie ich przenoszenie (wklejanie) do innego ekranu lub projektu. Aby z niego skorzystać, należy po prostu przeciągać i upuszczać bloki do plecaka. Z kolei, aby wkleić bloki, należy kliknąć na **Plecak**, a następnie przeciągnąć interesujące bloki do obszaru roboczego.

4. Polecenia pozwalające na **sprawdzanie poprawności składniowej** bloków, **wyśrodkowanie widoku** oraz **powiększenie** lub **pomniejszenie** obszaru roboczego.
5. **Kosz** – służy do usuwania bloków; można przeciągnąć na niego niepotrzebne bloki z przestrzeni roboczej.

Przedstawienie **bloków komponentowych** znacząco przekracza możliwości objętościowe niniejszych materiałów. We wcześniejszym podrozdziale, przy każdym komponencie, zamieszczano link do dokumentacji technicznej, gdzie można odnaleźć wszelkie potrzebne informacje, w ogólności: https://ai2.appinventor.mit.edu/reference/components/. Temat ten zostanie zilustrowany na przykładzie jednego z najczęściej używanych komponentów: przycisku o nazwie **Przycisk1** – rys. 28.

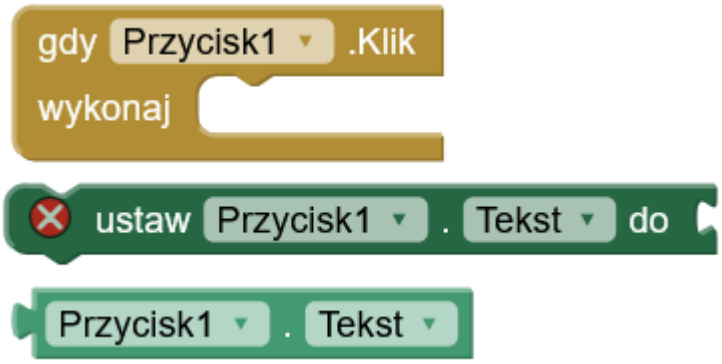
[Tekst alternatywny. Rysunek 28 obejmuje screen ekranu przedstawiający fragment narzędzia MIT App Inventor w trybie Edytor Blokowy z widocznymi przykładowymi blokami dla komponentu typu Przycisk.]



Rys. 28. Przykładowe, dostępne bloki dla komponentu typu Przycisk

Do obszaru roboczego zostały przeciągnięte trzy bloki widoczne na rys. 29. Żółty, **gdy Przycisk1.Klik wykonaj**, pozwala reagować na określone zdarzenie, tj. np. działanie użytkownika – tu, wszystko, co zostanie umieszczone wewnątrz tego bloku, będzie wykonywane w momencie kliknięcia przycisku. Ciemnozielony, **ustaw Przycisk1.Tekst do**, służy do zmiany wartości właściwości komponentu w trakcie działania aplikacji – tu pozwala zmienić tekst wyświetlany na przycisku; oczywiście wymagane jest podłączenie wartości tekstowej. Zielony, **Przycisk1.Tekst**, pozwala na pobranie aktualnej wartości właściwości komponentu – tu tekstu znajdującego się na przycisku. Należy także zwrócić uwagę na różnice w wyglądzie dwóch ostatnich bloków – każdy z nich ma inny kształt z lewej strony, który wskazuje jego rolę w programie. Blok pozwalający na odczyt właściwości posiada „wypustkę” z lewej

strony, co oznacza, że można go wstawić jako wartość do innego bloku. Z kolei blok pozwalający na ustawienie właściwości ma płaski, prostokątny początek (z zaokrąglonymi rogami), co wskazuje na to, że jest to blok wykonawczy.



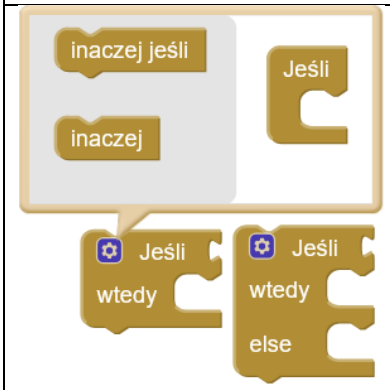
Rys. 29. Przykładowe bloki w obszarze roboczym dla komponentu typu Przycisk

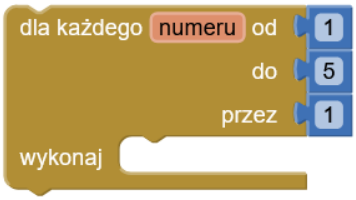
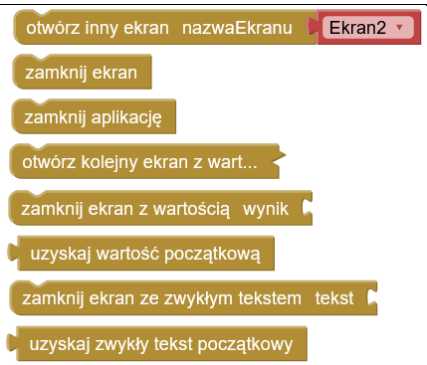
[Tekst alternatywny. Rysunek 29 przedstawia przykładowe trzy bloki dla komponentu typu Przycisk umieszczone w obszarze roboczym: żółty – gdy Przycisk1.Klik wykonaj, ciemnozielony – ustaw Przycisk1.Tekst do oraz zielony – Przycisk1.Tekst].

Bloki wbudowane pogrupowane są w 9 tematycznych kategorii. Dokładny opis dostępny jest na stronie: https://ai2.appinventor.mit.edu/reference/blocks/. Poniżej przedstawiono pokrótce wybrane z nich:

- **Kontrola** (ang. *Control*) – obejmuje bloki służące do sterowania przepływem programu, takie jak instrukcje warunkowe oraz pętle, a także do wywoływania różnych zdarzeń. Wybrane z nich omówiono w tabeli 8.

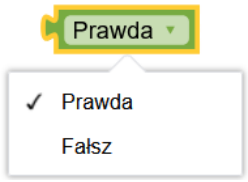
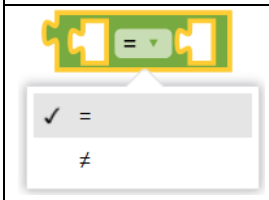
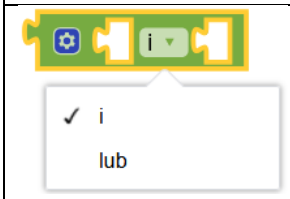
Tabela 8. Wybrane bloki z grupy Kontrola w narzędziu MIT App Inventor

Blok	Opis
	**Jeżeli** – podstawowa instrukcja warunkowa, która sprawdza, czy spełniony jest dany warunek. Jeśli warunek po **Jeśli** jest spełniony, to wykonane zostaną działania w sekwencji bloków **wtedy**. Natomiast jeśli warunek nie będzie spełniony, bloki będą zignorowane. Pełna postać: instrukcję warunkową można rozbudowywać w miarę potrzeb (do czego można wykorzystać przycisk – koło zębate na niebieskim tle). Jeżeli warunek po **Jeśli** jest spełniony, to wykonane zostaną działania w sekwencji bloków **wtedy**. Jeżeli jednak warunek nie zostanie spełniony, to program

Blok	Opis
	przechodzi do sprawdzenia warunku (lub kilku) **inaczej jeśli** i jeżeli warunek jest spełniony, wykonuje działania z odpowiedniej sekwencji bloków **wtedy**. Natomiast jeśli i ten warunek nie jest spełniony, wykonane zostaną działania z sekwencji bloków **else**.
	**Wykonaj dla każdej liczby od / do z krokiem** – pętla, która uruchamia bloki w sekcji **wykonaj** dla każdej wartości liczbowej z zakresu **od** … **do** z krokiem **przez**.
	**Wykonaj dla każdego elementu z listy** – pętla, która wykonuje działanie dla każdego elementu z listy.
	**While** – instrukcje w sekcji **wykonaj** wykonywane są tak długo, jak długo spełniany jest warunek logiczny w sekcji **gdy test**. Za każdym razem po wykonaniu instrukcji program sprawdza, czy warunek jest prawdziwy; kiedy warunek przestanie być spełniany, czyli wynikiem będzie **Fałsz**, pętla zatrzyma się.
	**Wykonaj i zwróć wynik** – blok ten może być wykorzystany w innej instrukcji, gdzie trzeba wykonać jakieś działanie i otrzymać w ten sposób wartość. Można również wykorzystać ten blok zamiast tworzenia nowej procedury.
	**Uruchom, ale zignoruj wynik** – uruchamia połączony blok kodu, jednakże ignoruje zwracaną przez niego wartość. Może to być przydatne w przypadku zdefiniowania procedury zwracającej wynik, ale wywołanej w kontekście, który nie akceptuje wyniku.
	**Ekrany / zamykanie aplikacji:** • Otwórz ekran o podanej nazwie. • Zamknij ekran. • Zamknij aplikację. • Otwórz inny ekran z ustaloną wartością początkową (tu jest zwinięty). • Zamknij ekran i zachowaj wartość (która będzie przekazana do kolejnego otwartego ekranu). • Otrzymaj wartość początkową (która została przekazana do tego ekranu). • Zamknij ekran i zwróć tekst do aplikacji (która go otworzyła). • Otrzymaj tekst początkowy (który był przekazany do ekranu przez aplikację).

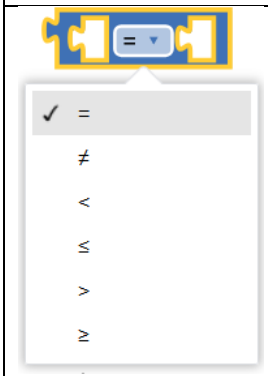
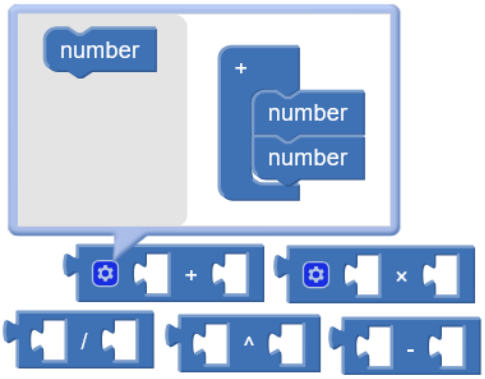
- **Logiczne** (ang. *Logic*) – zawiera bloki istotne z punktu widzenia podejmowania decyzji w programie, np. operatory logiczne i porównania. Wybrane z nich omówiono w tabeli 9.

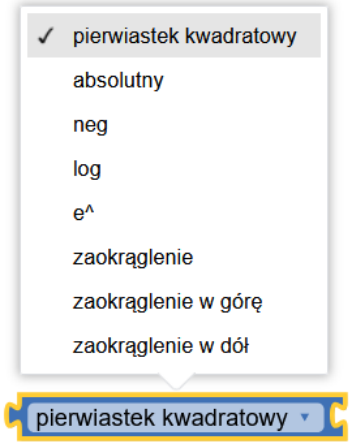
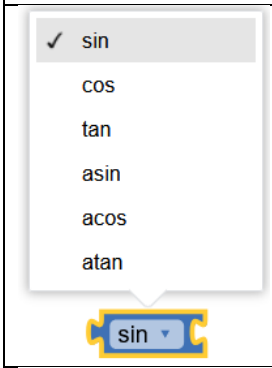
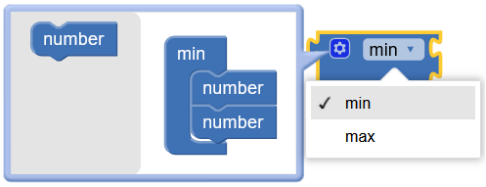
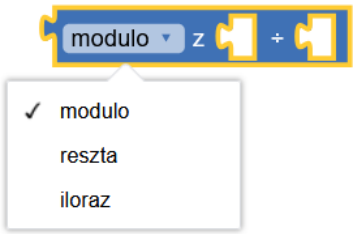
Tabela 9. Wybrane bloki z grupy Logiczne w narzędziu MIT App Inventor

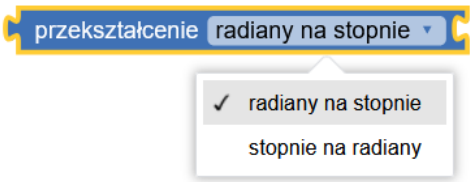
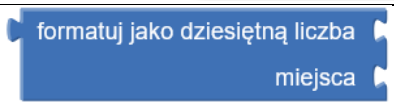
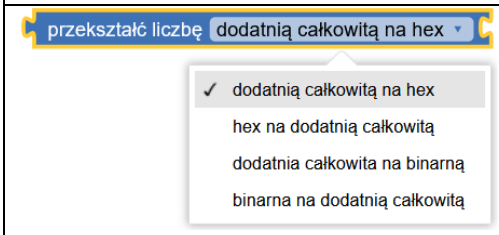
Blok	Opis
	**Podstawowy blok logiczny** – zwraca wartość logiczną **Prawda** lub **Fałsz**.
	**Not** – **negacja** (zaprzeczenie); np. zwraca wartość **Prawda**, gdy wejściowe dane są fałszywe.
	Sprawdza, czy wprowadzone argumenty są **równe** (**różne**): • dwie liczby są równe, gdy ich wartość liczbowa jest taka sama, np. 2 jest równe 2,00, • dwa bloki tekstowe są równe, gdy składają się z identycznych znaków w takiej samej kolejności (uwaga na duże litery), • liczby i tekst są równe, gdy wartość pewnej liczby odpowiada cyfrom uzyskanym z tekstu, • dwie listy są równe, gdy mają taką samą liczbę identycznych elementów. Działanie jest takie samo, jak odpowiadającego mu bloku w grupie bloków matematycznych.
	**i** – koniunkcja – sprawdza, czy wszystkie podane warunki logiczne są prawdziwe; rezultat przyjmuje wartość **Prawda** wtedy, gdy wszystkie składowe warunki są prawdziwe. **lub** – alternatywa – sprawdza, czy dowolny podany warunek logiczny jest prawdziwy; rezultat przyjmuje wartość **Prawda** wtedy, gdy co najmniej jeden składowy warunek jest prawdziwy.

- **Matematyczne** (ang. *Math*) – obejmuje bloki pozwalające na wykonywanie obliczeń oraz przetwarzanie danych liczbowych, np. dodawanie, odejmowanie, potęgowanie, pierwiastkowanie, funkcje trygonometryczne oraz losowanie liczb. Wybrane z nich omówiono w tabeli 10.

Tabela 10. Wybrane bloki z grupy Matematyczne w narzędziu MIT App Inventor

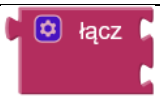
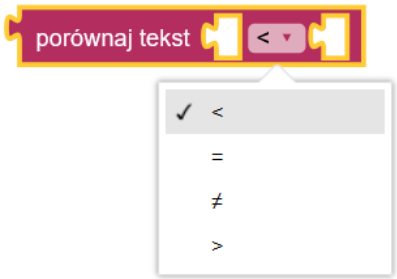
Blok	Opis
	**Blok numeryczny** – podstawowy blok przechowujący dowolną dodatnią lub ujemną wartość liczbową będącą liczbą dziesiętną.
	**Blok porównania dwóch wartości liczbowych** – przykładowo, blok widoczny po lewej stronie sprawdza, czy dwie wprowadzone wartości liczbowe są równe i zwraca wartość **Prawda** lub **Fałsz**. Należy pamiętać, że niektóre bloki są rozwijalne, co oznacza, że można je przekształcić w bloki wykonujące różne operacje. Dostępne operatory: • **=** – równe, • **≠** – różne, • **<** – mniejsze niż, • **≤** – mniejsze lub równe, • **>** – większe niż, • **≥** – większe lub równe.
	**Bloki działań arytmetycznych** – służą do wykonywania podstawowych operacji matematycznych na liczbach. Dostępne działania: • dodawanie (**+**) – sumuje dowolną liczbę wartości, • mnożenie (**×**) – oblicza iloczyn dowolnej liczby wartości, • odejmowanie (**−**) – odejmuje drugą wartość od pierwszej (dwa argumenty), • dzielenie (**/**) – dzieli pierwszą wartość przez drugą (dwa argumenty). • potęgowanie (**^**) – podnosi jedną liczbę (podstawę) do potęgi drugiej (wykładnika); również przyjmuje dwa argumenty.
	**Losowanie liczby całkowitej** z podanego zakresu (*od*, *do*).
	**Losowanie liczby z przedziału od 0 do 1**.

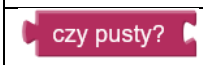
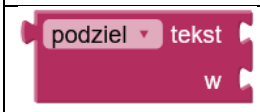
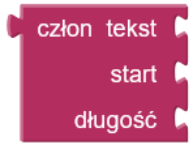
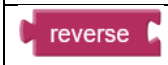
Blok	Opis
	**Blok operacji matematycznych** – wykonuje wybrane działanie matematyczne i oblicza: • **pierwiastek kwadratowy** podanej liczby, • **wartość bezwzględną** podanej liczby, • **ujemną wartość** podanej liczby, • **logarytm naturalny** podanej liczby, • **e** (e = 2,71828 …) **podniesioną do** podanej **potęgi**, • **zaokrągla** podaną liczbę **do najbliższej części całkowitej**, tj. jeśli część ułamkowa jest < ,5, zostanie zaokrąglona w dół, inaczej w górę, • **zaokrągla w górę** – zwraca najmniejszą liczbę całkowitą, która jest większa bądź równa od podanej liczby, • **zaokrągla w dół** – zwraca największą liczbę całkowitą, która jest mniejsza bądź równa od podanej liczby.
	**Funkcje trygonometryczne** – argument musi być wyrażony w stopniach. Wybrane operacje: • **sin** – zwraca wartość funkcji sinus, • **cos** – zwraca wartość funkcji cosinus. • **tan** – oblicza tangens kąta.
	**Min** / **max** – zwraca najmniejszą lub największą wartość z podanego zestawu liczb.
	Dzielenie dwóch liczb może dać wynik z częścią ułamkową (np. 5 / 2 = 2.5). **modulo** oraz **reszta** – reszta z dzielenia całkowitego; w przypadku liczb dodatnich wynik jest taki sam; modulo zachowuje znak dzielnika, a reszta znak dzielnej. **iloraz** – zwraca część całkowitą z dzielenia (odrzuca część dziesiętną wyniku).

Blok	Opis
	**Konwersja radianów na stopnie** – przekształca wartość kąta wyrażoną w radianach na stopnie. **Konwersja stopni na radiany** – przekształca wartość kąta podaną w stopniach na radiany.
	**Formatowanie liczby na dziesiętną** z podaną liczbą miejsc po przecinku – liczba miejsc po przecinku musi być nieujemną liczbą całkowitą
	**Czy obiekt jest liczbą?** – zwraca wartość **Prawda**, jeśli dany obiekt jest liczbą, a **Fałsz** w przeciwnym wypadku.
	**Konwersja wartości liczbowej** – konwertuje wartość liczbową pomiędzy dostępnymi systemami liczbowymi.

- **Tekst** (ang. *Text*) – zawiera bloki umożliwiające tworzenie oraz manipulację tekstem – m.in. łączenie, dzielenie, wyszukiwanie fragmentów, konwersję wielkości liter. Wybrane z nich omówiono w tabeli 11.

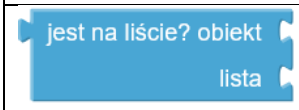
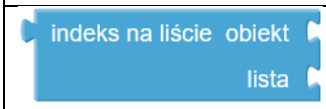
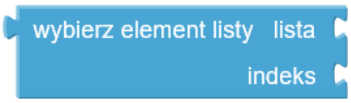
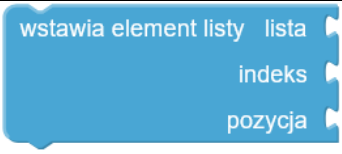
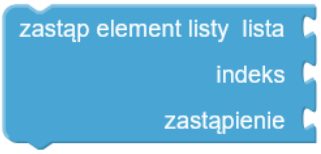
Tabela 11. Wybrane bloki z grupy Tekst w narzędziu MIT App Inventor

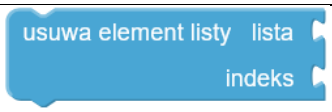
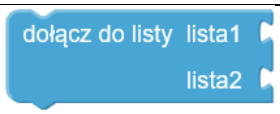
Blok	Opis
	**Ciąg tekstowy** – blok tekstowy zawierający dowolny ciąg znaków (litery, liczby lub inne znaki specjalne).
	**Połącz ciągi tekstowe** – łączy podane ciągi tekstowe (ang. *join*) w jeden.
	**Długość ciągu tekstowego** – zwraca liczbę znaków (razem ze spacjami) w podanym ciągu tekstowym.
	**Porównaj tekst** – porównuje dwa ciągi tekstowe w sposób leksykograficzny; istotna jest także wielkość liter (np. wielkie litery są uznawane za mniejsze niż małe litery). Dostępne operatory: • **<** – mniejsze niż, • **=** – równe, • **≠** – różne, • **>** – większe niż.

Blok	Opis
	**Czy pusty?** – sprawdza, czy podany ciąg znaków zawiera jakiekolwiek znaki – jeśli jest pusty, zwracana jest wartość **Prawda**, a w przeciwnym wypadku wartość **Fałsz**.
	**Przytnij** – usuwa z podanego ciągu znaków wszystkie spacje na początku i na końcu.
	**Podziel tekst** – dzieli podany ciąg tekstowy na części, wykorzystując do tego celu separator podany jako argument **w**; w wyniku zwracana jest lista *n*-elementowa.
	**Konwersja ciągu tekstowego**: • na **wielkie litery** – wszystkie litery zamieniane są na wielkie, • na **małe litery** – wszystkie litery zamieniane są na małe.
	**Czy zawiera?** **zawiera** – zwraca wartość **Prawda**, jeśli ciąg **fragment** pojawia się w ciągu tekstowym; w przeciwnym wypadku zwraca wartość **Fałsz**. **contains any** – zwraca wartość **Prawda**, jeśli którykolwiek z elementów na liście elementów (**piece list**) pojawia się w tekście; w przeciwnym wypadku zwraca wartość **Fałsz**. **contains all** – zwraca wartość **Prawda**, jeśli wszystkie elementy na liście elementów (**piece list**) pojawiają się w tekście; w przeciwnym razie zwraca wartość false.
	**Wyodrębnij człon tekstu** – wyodrębnia tekst o pożądanej **długości** z podanego ciągu tekstowego, począwszy od podanej **pozycji** (gdzie 1 oznacza początek).
	**Czy obiekt jest tekstem?** – zwraca wartość **Prawda**, jeśli dany obiekt jest typu tekstowego, a **Fałsz** w przeciwnym wypadku
	**Odwróć tekst** – odwraca podany tekst. Np. kot → tok.

- **Listy** (ang. *Lists*) – zawiera bloki pozwalające na pracę ze strukturami danych w postaci list, m.in. tworzenie, dodawanie oraz usuwanie elementów, wyszukiwanie, sortowanie, czy też pobieranie konkretnych pozycji. Wybrane z nich omówiono w tabeli 12.

Tabela 12. Wybrane bloki z grupy Listy w narzędziu MIT App Inventor

Blok	Opis
	**Utwórz pustą listę** – tworzy pustą listę bez elementów.
	**Twórz listę** – tworzy listę z dowolną liczbą elementów (które stanowią podpięte bloki). Jeśli żaden blok nie zostanie podpięty, to lista będzie pusta.
	**Dodaj elementy do listy** – dodaje nowy element(y) (**pozycja**) do istniejącej listy (**lista**) na jej koniec.
	**Jest na liście?** – sprawdza, czy dany element (**obiekt**) znajduje się w podanej liście.
	**Długość listy** – zwraca liczbę elementów znajdujących się na liście.
	**Czy lista pusta?** lista – sprawdza, czy lista jest pusta, tj. nie zawiera żadnych elementów (wtedy zwraca **Prawda**).
	**Wybierz losowy element** – zwraca losowy element z podanej listy.
	**Indeks na liście** obiekt lista – zwraca numer pozycji (indeksu) wskazanego obiektu (elementu) na liście; jeśli podana lista go nie zawiera, zwraca 0.
	**Wybierz element listy** – zwraca element znajdujący się na liście pod podanym numerem indeksu.
	**Wstawia element listy** – wstawia nowy element (**pozycja**) do listy (**lista**) w określonym przez numeru indeksu miejscu (**indeks**).
	**Zastąp element listy** – zamienia element znajdujący się na liście o podanym numerze indeksu na inny (**zastąpienie**).

Blok	Opis
	**Usuwa element** – usuwa element z podanej listy, znajdujący się pod określonym numerem indeksu.
	**Dołącz do listy** – dodaje wszystkie elementy z drugiej listy (**lista2**) na koniec pierwszej listy (**lista1**).

- **Dictionaries** – grupa bloków służących do obsługi **słowników**, czyli struktur danych składających się z par: klucz – wartość. Więcej: https://ai2.appinventor.mit.edu/reference/blocks/dictionaries.html.

- **Kolory** (ang. *Colors*) – obejmuje bloki pozwalające na wybieranie, definiowanie i manipulowanie kolorami. Wybrane z nich omówiono w tabeli 13.

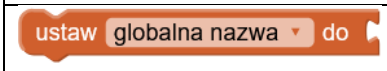
Tabela 13. Wybrane bloki z grupy Kolory w narzędziu MIT App Inventor

Blok	Opis
	**Podstawowy blok koloru** – prostokąt znajdujący się w środku reprezentuje przechowywaną barwę; po kliknięciu można wybrać jeden z predefiniowanych kolorów.
	**Tworzenie koloru** – blok pozwala na utworzenie koloru poprzez podanie wartości w systemie RGB (R – *Red*, G – *Green*, B – *Blue*). Czwarta, opcjonalna składowa odpowiada za przezroczystość.

- **Zmienne** (ang. *Variables*) – zawiera bloki umożliwiające tworzenie i modyfikację zmiennych globalnych oraz lokalnych, dzięki czemu można przechowywać dane, które zmieniają się w czasie działania aplikacji. Wybrane z nich omówiono w tabeli 14.

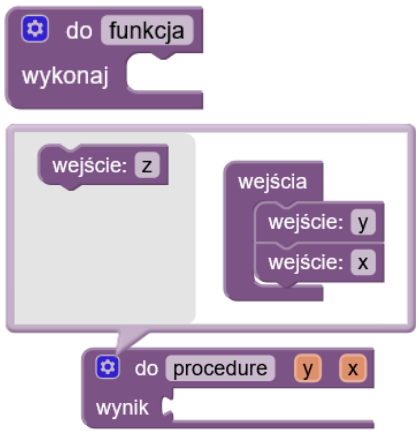
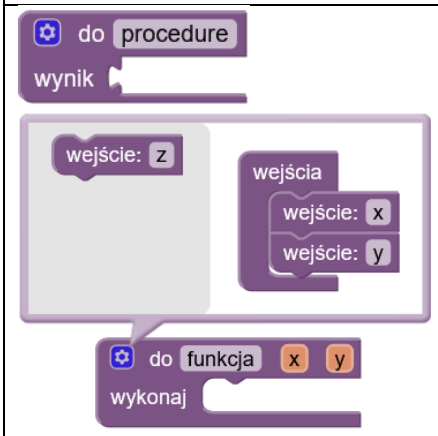
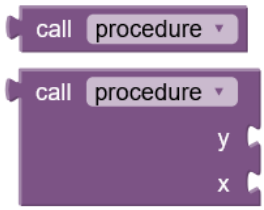
Tabela 14. Wybrane bloki z grupy Zmienne w narzędziu MIT App Inventor

Blok	Opis
	**Inicjowanie zmiennej globalnej** – tworzenie nowej zmiennej globalnej o podanej nazwie; można przypisać jej wartość początkową poprzez dołączenie odpowiedniego

Blok	Opis
	bloku. Zmienne takie są wykorzystywane we wszystkich procedurach i zdarzeniach, a ich wartości mogą się zmieniać w trakcie działania programu. Można zmienić nazwę zmiennej globalnej w dowolnym momencie – wszelkie powiązane bloki odnoszące się do starej nazwy zostaną automatycznie zaktualizowane.
	**Nadaj wartość zmiennej** – umożliwia nadanie nowej wartości dla zmiennej o podanej nazwie poprzez dołączenie odpowiedniego bloku.
	**Pobierz wartość** – pozwala na pobranie wartości istniejącej zmiennej.
	**Zainicjuj zmienną lokalną i wykonaj** – blok będący mutatorem, umożliwiający tworzenie nowych zmiennych, które są używane tylko w procedurze uruchamianej w części *DO* bloku. W ten sposób wszystkie zmienne w tej procedurze będą rozpoczynać się od tej samej wartości za każdym razem, gdy procedura zostanie uruchomiona.
	**Zainicjuj zmienną lokalną i zwróć** – blok będący mutatorem, umożliwiający tworzenie nowych zmiennych, które są używane tylko w procedurze uruchamianej w części *RETURN* bloku. W ten sposób wszystkie zmienne w tej procedurze będą rozpoczynać się od tej samej wartości za każdym razem, gdy procedura zostanie uruchomiona. Jest to blok zwracający wynik.

- **Funkcje** (ang. *Procedures*) – pozwala tworzyć podprogramy, które można wielokrotnie wykorzystywać. W informatyce **podprogram** to wydzielona część programu, wykonująca pewne operacje, którą można wielokrotnie wywoływać. Stosuje się je, aby uprościć program główny oraz zwiększyć czytelność kodu. W językach programowania podprogramy najczęściej dzielą się na: **funkcje** (które wykonują obliczenia i zwracają jakąś wartość) oraz **procedury** (nie zwracają żadnej wartości, chyba że poprzez odpowiednie parametry, a zamiast tego wykonują pewne działania). Należy jednakże zaznaczyć, że obecnie podział między procedurami, a funkcjami w wielu językach programowania jest czysto teoretyczny. W MIT App Inventor przez **procedurę** można rozumieć sekwencję bloków zapisaną pod pewną nazwą, którą można wielokrotnie wywoływać przy wykorzystaniu jednego bloku. Wybrane bloki z tej grupy omówiono w tabeli 15.

Tabela 15. Wybrane bloki z grupy Funkcje w narzędziu MIT App Inventor

Blok	Opis
	**Procedura wykonująca** (ang. *procedure do*) – blok tworzący podprogram będący procedurą, który wykonuje określoną sekwencję poleceń, ale nie zwraca żadnej wartości. Można przekazywać do niego argumenty, do czego należy wykorzystać przycisk – koło zębate na niebieskim tle. Nazwy procedur w aplikacji muszą być unikalne; można zmieniać je w dowolnym momencie – wszelkie powiązane bloki odnoszące się do starej nazwy zostaną automatycznie zaktualizowane.
	**Procedura zwracająca rezultat** (ang. *procedure result*) – blok tworzący podprogram będący funkcją, który wykonuje operacje i zwraca wynik (rezultat). Można przekazywać do niego argumenty, do czego należy wykorzystać przycisk – koło zębate na niebieskim tle. Nazwy procedur w aplikacji muszą być unikalne; można zmieniać je w dowolnym momencie – wszelkie powiązane bloki odnoszące się do starej nazwy zostaną automatycznie zaktualizowane.
	Blok pozwalający na **wywołanie** (uruchomienie) **procedury wykonującej** o podanej nazwie (opcjonalnie z argumentami).
	Blok pozwalający na **wywołanie** (uruchomienie) **procedury zwracającej rezultat** o podanej nazwie. Blok ten należy podłączyć do innego bloku oczekującego na zwróconą wartość. Rezultat (a więc wynik) wykonania tej procedury jest przekazywany do bloku, do którego jest podpięte to wywołanie procedury.

2.3.4. Aplikacje responsywne

Bardzo istotną, ale jednocześnie trudną i złożoną kwestią w procesie projektowania aplikacji mobilnych, jest tworzenie takich aplikacji, które będą wyglądały dobrze na urządzeniach z ekranami o różnych rozmiarach – np. zarówno na telefonie, jak i na tablecie. Powinno więc tworzyć się programy, które będą dostosowywać się do

różnych rozmiarów i rozdzielczości ekranów – jest to tzw. **projektowanie responsywne**.

W ogólności, powszechnym podejściem w zakresie projektowania responsywnego aplikacji mobilnych jest tworzenie programów, które zawierają dużo układów i wiele plików graficznych, aby jak najlepiej mogły dostosować się do różnych rozmiarów ekranu i rozdzielczości, co wymaga jednakże dużo pracy od deweloperów.
W narzędziu MIT App Inventor wykorzystywane jest prostsze, choć nieco bardziej ograniczone podejście:
https://ai2.appinventor.mit.edu/reference/other/responsiveDesign.html.

W MIT App Inventor można zdefiniować właściwość ekranu o nazwie **Rozmiar** (ang. *Sizing*) dla ekranu głównego aplikacji (**Screen1**) w zakładce **Project Properties** – por. rys. 14. Dostępne są dwie wartości:

- **Ustalony** (ang. *fixed*) – MIT App Inventor tworzy aplikacje dla ekranu o szerokości 320 pikseli i wysokości 460 pikseli, a następnie skaluje wygląd aplikacji, aby dopasować ją do rzeczywistego rozmiaru ekranu, na którym aplikacja jest uruchomiona.
- **Responsywny** (ang. *responsive*) – opcja zalecana; aplikacje utworzone przez MIT App Inventor używają rzeczywistej liczby pikseli dla urządzenia, na którym działa aplikacja. Wartość ta może być odmienna dla różnych urządzeń. Zasada podczas używania App Inventor do tworzenia aplikacji z responsywnym designem: należy określać **szerokości** i **wysokości** komponentów jako wartości procentowe szerokości i wysokości ekranu, a nie jako stałe liczby pikseli – por. rys. 23. Na przykład, aby utworzyć przycisk o szerokości równej połowie szerokości ekranu, należy ustawić jego **szerokość** na 50%, a nie na określoną liczbę pikseli. "Procent" oznacza tutaj procent szerokości lub wysokości ekranu, a nie procent elementu zawierającego.

Uwaga: Na obecną chwilę problematyczne, w zakresie projektowania aplikacji responsywnych, może być tworzenie rysunków i animacji przy wykorzystaniu komponentów z grupy **Rysowanie i animacja**. W przypadku płótna (**Canvas**), bardzo często do tworzenia rysunków i kontrolowania interakcji używa się wartości współrzędnych, co może być skomplikowane w przypadku obsługi wielu różnych rozmiarów ekranu. W ogólności, można w ogóle nie wykorzystywać tutaj projektu responsywnego lub skorzystać z podejścia „mieszanego” (tj. stały rozmiar płótna, a reszta aplikacji responsywna), jednak najlepsze rezultaty, choć pracochłonne i wymagające kilkunastu dodatkowych bloków, przyniesie ustawianie rozmiarów

komponentów z grupy **Rysowanie i animacja** podczas uruchamiania aplikacji na odpowiednie wartości w odniesieniu do rzeczywistego rozmiaru ekranu.

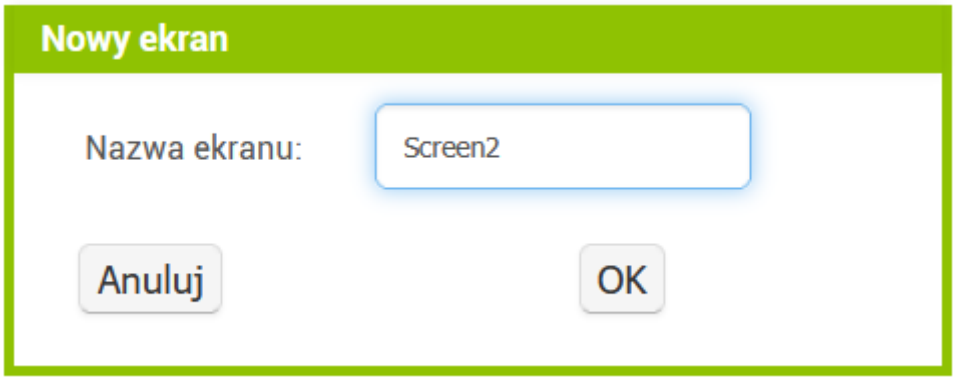
2.3.5. Aplikacje wieloekranowe

Ekran to komponent najwyższego poziomu zawierający wszystkie pozostałe komponenty programu. Więcej na: http://ai2.appinventor.mit.edu/reference/components/userinterface.html#Screen.

W narzędziu MIT App Inventor, ekrany służą do organizowania tworzonej aplikacji mobilnej w oddzielne, niezależne części funkcjonalne. Dzięki temu można tworzyć tzw. **aplikacje wieloekranowe**, w których każdy ekran ma swój własny wygląd i logikę działania.

Bardzo istotne jest to, że **Screen1** to zawsze pierwszy i jednocześnie główny (Startowy) ekran aplikacji – jest tworzony automatycznie i zawsze musi istnieć.

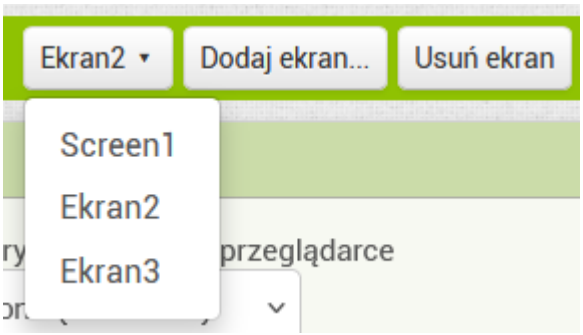
Można oczywiście dodawać kolejne ekrany (np. **Screen2**, **Screen3** – jednak nie więcej niż 10), aby oddzielić różne funkcje aplikacji – np. ekran logowania, ekran ustawień, ekran gry itd. W tym celu należy nacisnąć przycisk **Dodaj ekran …** w trybie **Projektant** – por. rys. 13; pojawi się nowe okno widoczne na rys. 30, gdzie należy podać nazwę tego ekranu. Ekrany nie działają jednocześnie – aplikacja może wyświetlać tylko jeden ekran w danym momencie.



Rys. 30. Okno – Nowy ekran

[Tekst alternatywny. Rysunek 30 przedstawia fragment narzędzia MIT App Inventor: okno pozwalające na dodanie nowego ekranu, gdzie trzeba podać jego nazwę (tu wpisano Screen2).]

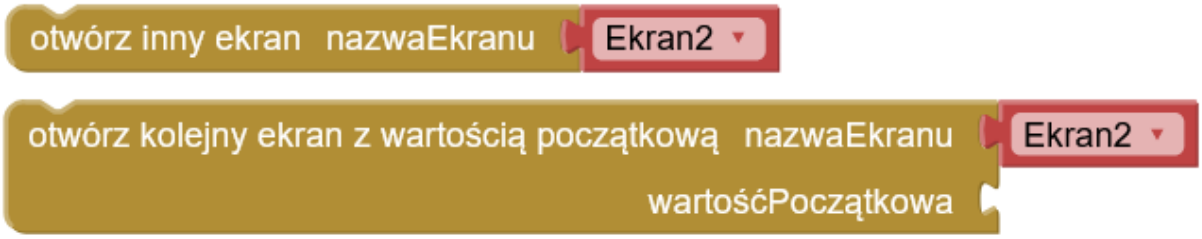
W trybie **Projektant** można przełączać się pomiędzy ekranami – rys. 31.



Rys. 31. Przełączanie się pomiędzy ekranami w trybie Projektant

[Tekst alternatywny. Rysunek 31 obejmuje fragment narzędzia MIT App Inventor w trybie Projektant, prezentujący możliwość przełączania pomiędzy ekranami.]

W **Edytorze Blokowym**, do otwierania innych ekranów używa się specjalnych bloków z grupy **Kontrola** – rys. 32.



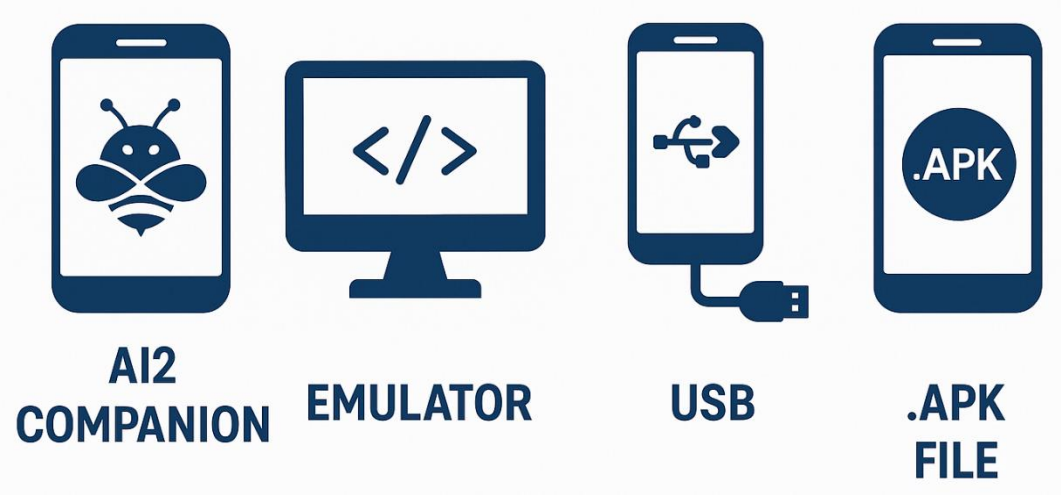
Rys. 32. Przykładowe bloki do otwierania innych ekranów

[Tekst alternatywny. Rysunek 32 przedstawia przykładowe bloki do otwierania innych ekranów, dostępne w Edytorze Blokowym w grupie Kontrola.]

Więcej informacji na temat aplikacji wieloekranowych można odnaleźć na: https://ai2.appinventor.mit.edu/reference/other/manyscreens.html.

2.4. Testowanie i uruchamianie aplikacji

W MIT App Inventorze budowane są aplikacje na urządzenia mobilne – efekty pracy nie są więc widoczne bezpośrednio w tym narzędziu. Aby śledzić postępy prac, należy wykorzystać fizyczne urządzenie (np. tablet bądź telefon z systemem Android lub iOS) albo wbudowany do programu MIT App Inventor emulator – rys. 33. Aktualne informacje można przeczytać na stronie: http://appinventor.mit.edu/explore/ai2/setup.html.



Rys. 33. Możliwości testowania aplikacji w narzędziu MIT App Inventor (grafikę wygenerowano z wykorzystaniem generatywnej sztucznej inteligencji w ramach licencji użytkownika)

[Tekst alternatywny. Rysunek 33 przedstawia cztery główne sposoby testowania aplikacji utworzonych poprzez narzędzie MIT App Inventor: połączenie z urządzeniem mobilnym przez Internet, użycie emulatora na komputerze, połączenie przez kabel USB oraz instalację pliku APK. Każdy sposób zilustrowano ikonami symbolizującymi te urządzenia.]

Chcąc zatem przetestować działanie aplikacji, należy wybrać jedną z dostępnych opcji, co uzależnione jest szczególnie od faktu posiadania odpowiedniego fizycznego urządzenia mobilnego:

- Testowanie aplikacji w czasie rzeczywistym za pomocą fizycznego urządzenia mobilnego (smartfon lub tablet z systemem Android, iPhone, iPad) wraz z zainstalowanym oprogramowaniem **MIT AI2 Companion** przy wykorzystaniu sieci Internet. W momencie tworzenia niniejszych materiałów, w dokumentacji można odnaleźć informację, że muszą być one podłączone do tej samej sieci Wi-Fi, jednakże komunikacja działa także, gdy to założenie nie jest spełnione (np. połączenie przy wykorzystaniu transmisji sieci komórkowej) – http://appinventor.mit.edu/explore/ai2/setup-device-wifi.
- Testowanie aplikacji w czasie rzeczywistym przy wykorzystaniu emulatora zainstalowanego na komputerze – http://appinventor.mit.edu/explore/ai2/setup-emulator.
- Testowanie aplikacji w czasie rzeczywistym za pomocą urządzenia mobilnego z systemem Android wraz z zainstalowanym oprogramowaniem **MIT AI2**

Companion przy wykorzystaniu kabla USB – http://appinventor.mit.edu/explore/ai2/setup-device-usb.

- Zbudowanie aplikacji (na obecną chwilę dostępne w pełni ***.apk** dla systemu Android, a także ***.ipa** – lecz w wersji beta) i zainstalowanie jej na fizycznym urządzeniu.

Uwaga: Jak wspomniano już wcześniej, współpraca z urządzaniami mobilnymi z systemem iOS dostępna jest od marca 2021 i jest stale rozwijana (nie wszystkie funkcjonalności działają na obecną chwilę, lecz **sytuacja jest dynamiczna**) – więcej można przeczytać na stronie: https://appinventor.mit.edu/ios_tips.

3. Case study – responsywna aplikacja wieloekranowa

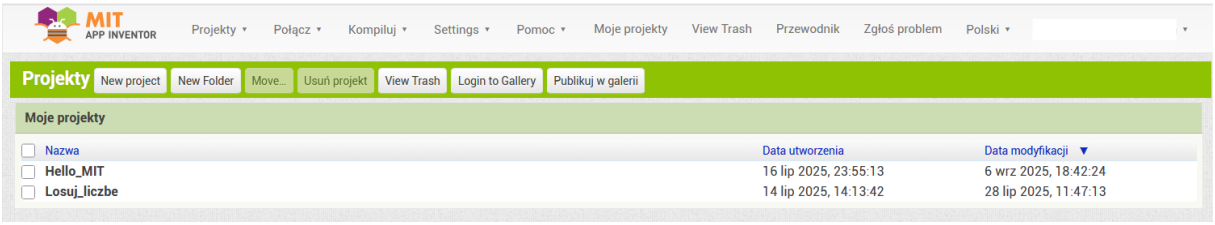
W poprzednich rozdziałach przedstawiona została idea tworzenia aplikacji mobilnych przy zastosowaniu narzędzi **no-code** / **low-code**, wraz z dość szczegółowym omówieniem wybranego z nich: **MIT App Inventor**. Teraz nadszedł czas, aby wykorzystać zdobytą wiedzę w praktyce i wykonać przykładową, wieloekranową aplikację mobilną o nazwie **Hello_ID**, a następnie przetestować jej działanie.

Aplikacja **Hello_ID** ma składać się z 3 ekranów o poniższych funkcjonalnościach, a nawigacja ma odbywać się poprzez menu główne:

- Menu główne – ekran **Screen1**, wraz z dodatkowymi przyciskami: **Autor** (po naciśnięciu ma zostać wyświetlony komunikat typu ***MessageDialog*** z informacjami o Autorze) oraz **Zamknij** (kończy pracę aplikacji),
- Losowanie liczby z zakresu od 1 do 10 – ekran **Losowanie**; losowa liczba ma pojawiać się po otwarciu ekranu, a następnie po potrząśnięciu urządzeniem lub po kliknięciu przycisku.
- Wykreślenie przykładowego wykresu – ekran **Wykres**.

3.1. Logowanie i utworzenie nowego projektu

Na początek, zgodnie z informacjami zamieszczonymi w podrozdziale 2.2., należy zalogować się do narzędzia MIT App Inventor. Jeżeli Użytkownik pracował już nad jakimś projektem, zostanie on otwarty – należy więc go zamknąć i przejść do widoku swoich projektów (rys. 34; **Projekty → Moje projekty**).

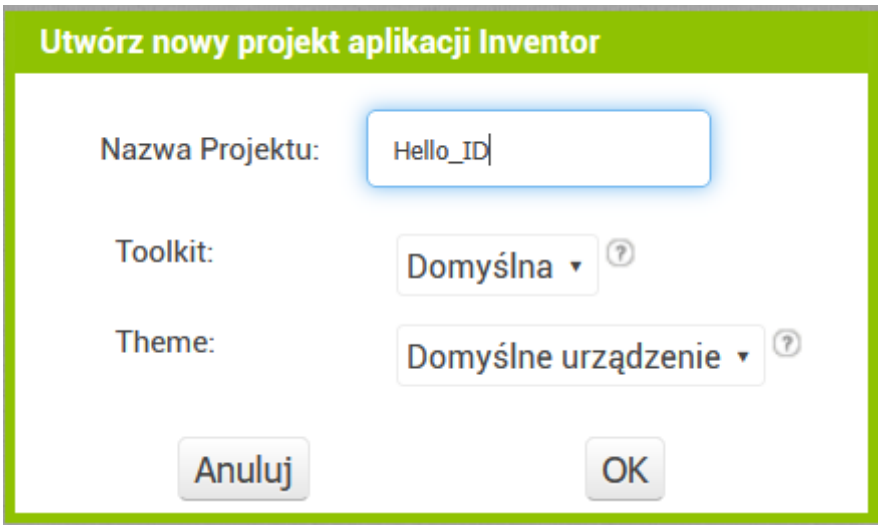


Rys. 34. Fragment zakładki Projekty w narzędziu MIT App Inventor

[Tekst alternatywny. Rysunek 34 przedstawia screen ekranu, na którym znajduje się górny fragment zakładki Projekty w narzędziu MIT App Inventor. Widoczne są dwa projekty: Hello_MIT oraz Losuj_liczbe.]

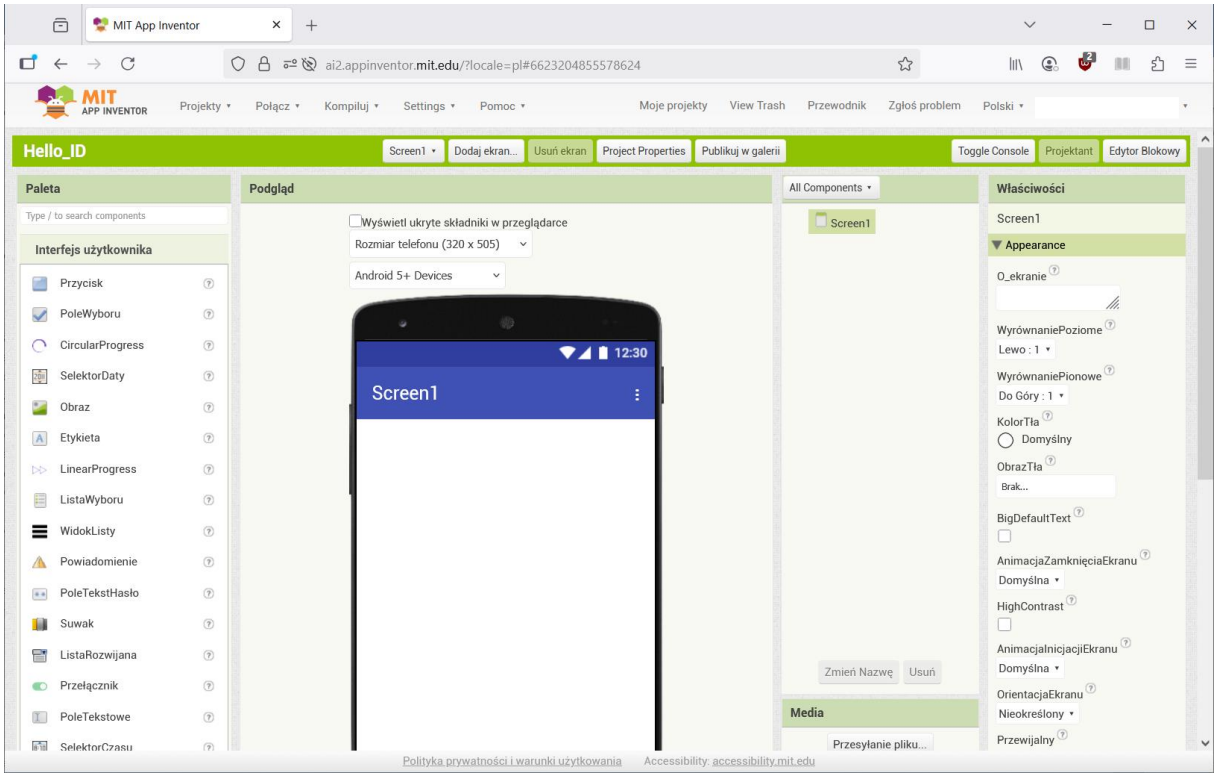
Następnie trzeba utworzyć nowy projekt; w tym celu należy wybrać polecenie **New project** – pojawi się nowe okno, w którym należy podać jego nazwę: **Hello_ID** – rys.

35. Pozostałe opcje należy pozostawić bez zmian i nacisnąć przycisk **OK**. Stworzony projekt otworzy się w trybie **Projektant** – rys. 36.



Rys. 35. Tworzenie nowego projektu w MIT App Inventor o nazwie Hello_ID

[Tekst alternatywny. Rysunek 35 przedstawia okno tworzenia nowego projektu w narzędziu MIT App Inventor. Podano nazwę Hello_ID.]

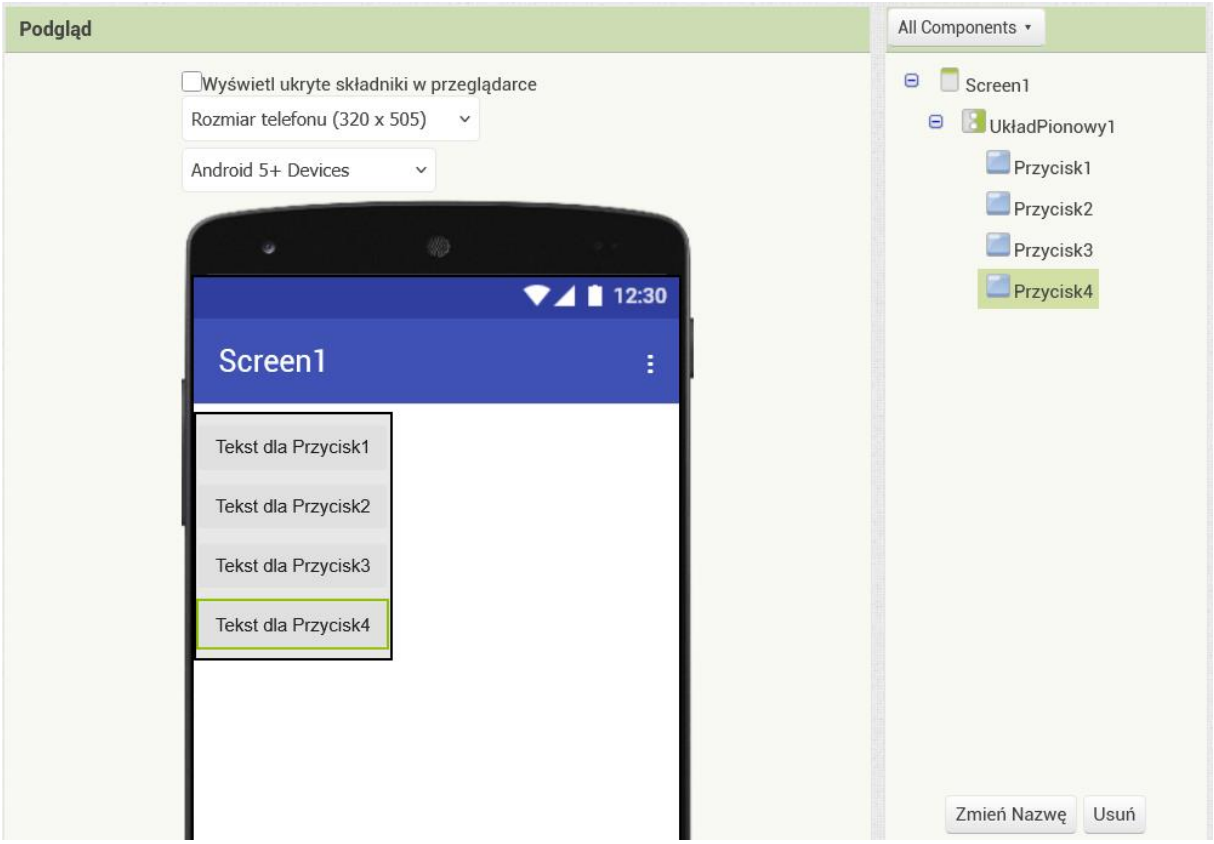


Rys. 36. Nowoutworzona aplikacja Hello_ID w trycie Projektant

[Tekst alternatywny. Rysunek 36 przedstawia screen ekranu z przeglądarki internetowej Mozilla Firefox – nowoutworzoną aplikację Hello_ID w trybie Projektant.]

3.2. Menu główne. Dodawanie ekranów

Po uruchomieniu aplikacji użytkownik ma zobaczyć menu główne (ekran **Screen1**), które będzie służyć do wyboru żądanej funkcjonalności – tj. uruchomienia innego ekranu, wyświetlenia komunikatu z informacjami o Autorze lub zamknięcia aplikacji. W tym celu należy wykorzystać kilka komponentów: odpowiedni układ (komponent **UkładPionowy** z grupy **Układ**) oraz 4 przyciski (komponenty typu **Przycisk** z grupy **Interfejs użytkownika**); elementy te należy przeciągnąć z **Palety** znajdującej się po lewej stronie, do obszaru wirtualnego telefonu tak, aby wszystkie **Przyciski** znajdowały się wewnątrz **UkładuPionowego** (struktura hierarchiczna) – rys. 37.

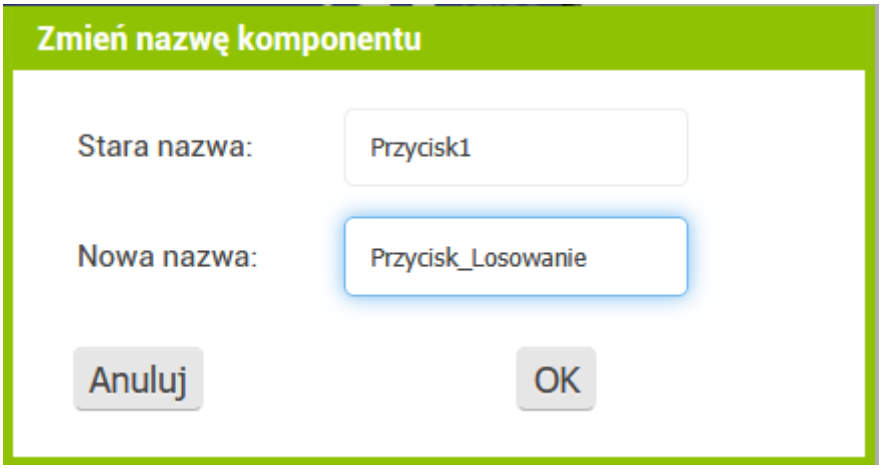


Rys. 37. Nowoutworzona aplikacja Hello_ID w trycie Projektant

[Tekst alternatywny. Rysunek 37 przedstawia screen ekranu, na którym znajduje się fragment narzędzia MIT App Inventor w trybie Projektant. Widać obszar wirtualnego

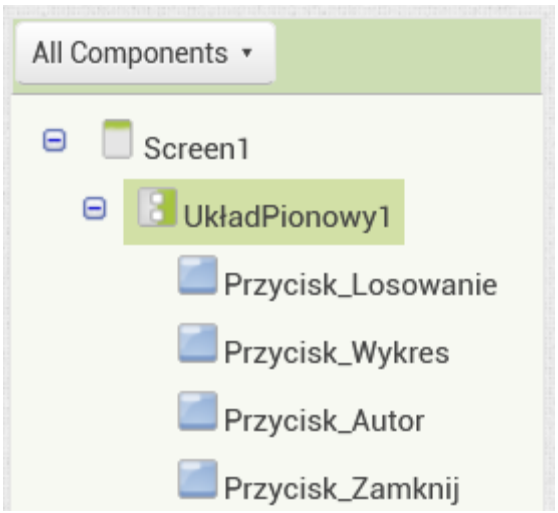
telefonu, a po prawej stronie nazwy komponentów: Screen1, UkładPionowy1 i 4 przyciski.]

Następnie wykorzystując opcję **Zmień Nazwę**, można zmienić nazwy komponentów typu **Przycisk** na bardziej czytelne, np. na: **Przycisk_Losowanie** (rys. 38), **Przycisk_Wykres**, **Przycisk_Autor** oraz **Przycisk_Zamknij**. W tym celu, każdorazowo, należy podać nową nazwę i nacisnąć przycisk **OK**. Stan po nadaniu nowych nazw przedstawia rys. 39.



Rys. 38. Nowoutworzona aplikacja Hello_ID w trycie Projektant

[Tekst alternatywny. Rysunek 38 przedstawia fragment narzędzia MIT App Inventor – okno pozwalające na zmianę nazwy komponentu (tu: Przycisk1 na Przycisk_Losowanie).]



Rys. 39. Nazwy komponentów po dokonanych zmianach

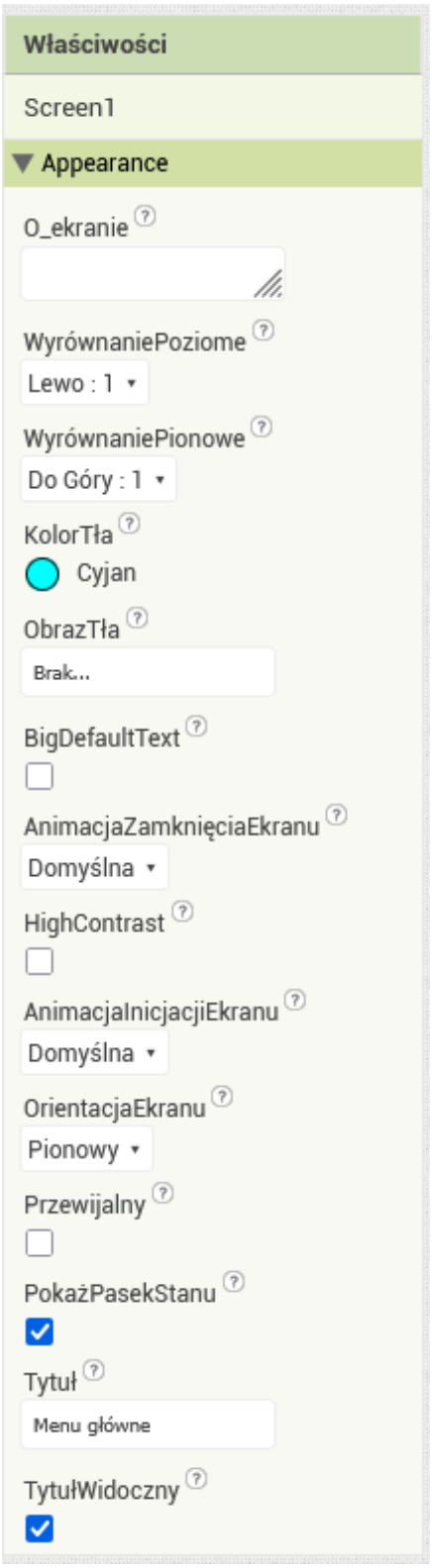
[Tekst alternatywny. Rysunek 39 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są nazwy komponentów: Screen1, UkładPionowy1 i 4 przyciski: Przycisk_Losowanie, Przycisk_Wykres, Przycisk_Autor oraz Przycisk_Zamknij.]

W kolejnym kroku należy zmienić wybrane właściwości komponentów:

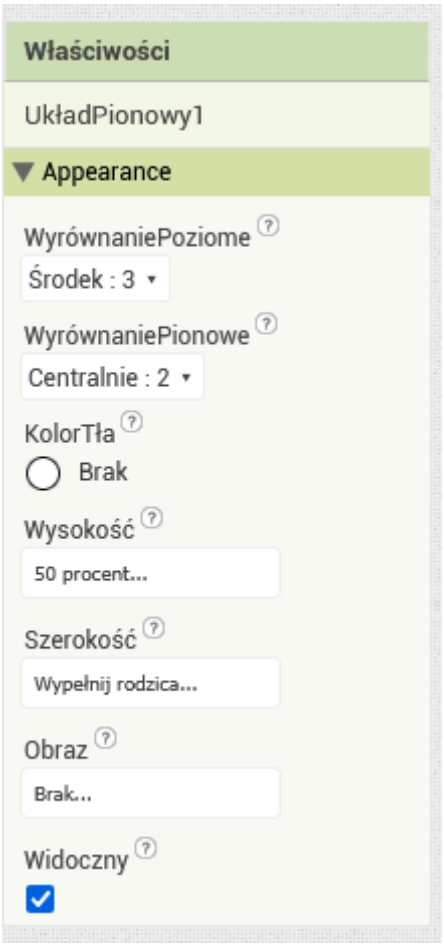
a) Należy zapoznać się z właściwościami projektu w oknie **Project Properties** (por. rys. 14), w szczególności, czy **Rozmiar** (zakładka **General**) jest ustawiony na **Responsywny**. Można tu także zmienić ikonę aplikacji.
b) Dla komponentu **Screen1** ustawić (rys. 40; reszta – ustawienia domyślne):
    a. **KolorTła** – **Cyjan**,
    b. **OrientacjaEkranu** – **Pionowy**; ekran nie będzie się obracał,
    c. **Tytuł** – wpisać: **Menu główne**.
c) Dla komponentu **UkładPionowy1** ustawić (rys. 41; reszta – ustawienia domyślne):
    a. **WyrównaniePoziome**: **Środek**,
    b. **WyrównaniePionowe**: **Centralnie**,
    c. **KolorTła**: **Brak**,
    d. **Wysokość**: **50 procent**; dzięki temu aplikacja będzie responsywna – wysokość tego komponentu, obejmującego cztery przyciski, będzie zawsze zajmować połowę ekranu,
    e. **Szerokość**: **Wypełnij rodzica**.

[Tekst alternatywny. Rysunek 40 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu Screen1.]

[Tekst alternatywny. Rysunek 41 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu UkładPionowy1.]

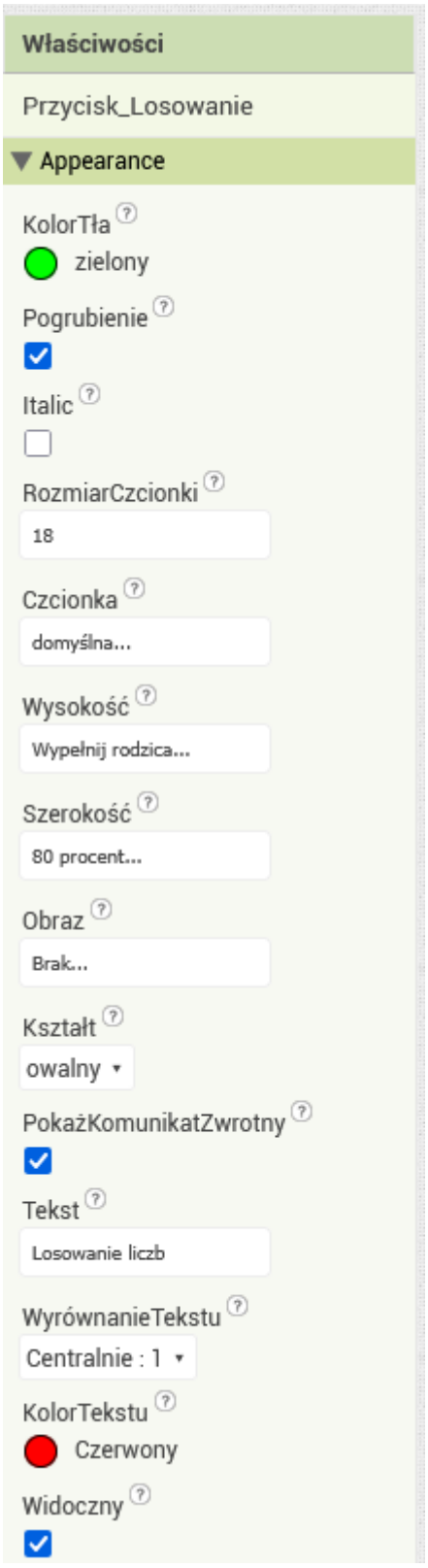


Rys. 40. Właściwości dla komponentu Screen1



Rys. 41. Właściwości dla komponentu UkładPionowy1

a) Dla wszystkich komponentów typu **Przycisk** ustawić (reszta – ustawienia domyślne):
   a. **KolorTła**: **Zielony**,
   b. **Pogrubienie**: **Tak**,
   c. **RozmiarCzcionki**: **18**,
   d. **Wysokość**: **Wypełnij rodzica**,
   e. **Szerokość**: **80 procent**,
   f. **Kształt**: **Owalny**,
   g. **KolorTekstu**: **Czerwony**
   h. Odpowiedni tekst na każdym przycisku: **Przycisk_Losowanie** (rys. 42) – Losowanie liczb, **Przycisk_Wykres** – Stwórz wykres, **Przycisk_Autor** – Informacje o Autorze, **Przycisk_Zamknij** – Zamknij aplikację.



Rys. 42. Właściwości dla komponentu Przycisk_Losowanie

[Tekst alternatywny. Rysunek 42 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu Przycisk_Losowanie.]

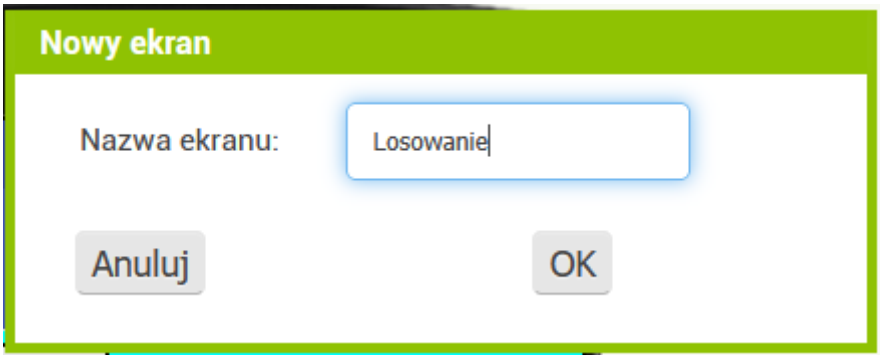
Oczekiwany wygląd ekranu głównego (**Screen1**) przedstawiono na rys. 43.

Rys. 43. Oczekiwany wygląd ekranu głównego (Screen1)

[Tekst alternatywny. Rysunek 43 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant – wirtualny telefon. Zaprezentowany jest oczekiwany wygląd

ekranu głównego (Screen1), m.in. kolor tła cyjan i cztery owalne przyciski (zielony kolor tła, czerwony tekst).]

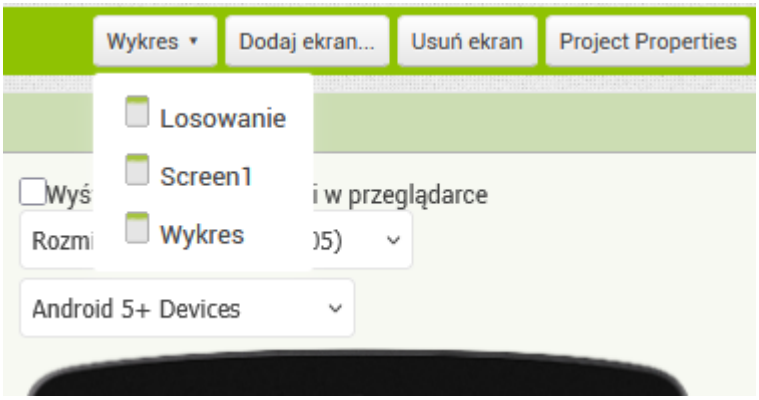
W kolejnym kroku, korzystając z opcji **Dodaj ekran**, należy dodać 2 nowe ekrany: **Losowanie** (rys. 44) i **Wykres** – dla każdego z nich, zarówno wygląd, jak i „kod" będą tworzone odrębnie.



Rys. 44. Okno Nowy ekran – tworzenie ekranu Losowanie

[Tekst alternatywny. Rysunek 44 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant – okno Nowy ekran, które pozwala na utworzenie nowego ekranu (tu o nazwie Losowanie).]

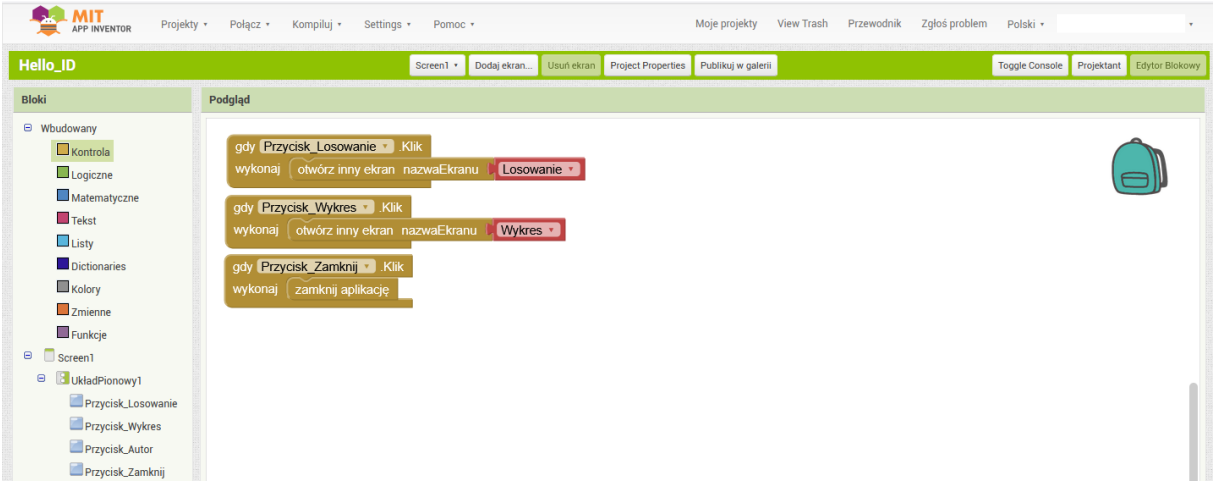
Pomiędzy utworzonymi ekranami można się przełączać – rys. 45.



Rys. 45. Przełączanie się pomiędzy ekranami

[Tekst alternatywny. Rysunek 45 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant – możliwość przełączania się pomiędzy ekranami. Widoczne są 3 ekrany: Losowanie, Screen1 oraz Wykres.]

Po zdefiniowaniu wyglądu ekranu **Screen1**, a także dodaniu dwóch pozostałych ekranów, należy przejść do trybu **Edytor Blokowy** dla ekranu **Screen1** i utworzyć odpowiednie bloki kodu; należy przypisać każdemu przyciskowi odpowiednie działanie (tj. otwarcie nowego ekranu, wyświetlenie komunikatu lub zakończenie aplikacji). Poniżej (rys. 46) zaprezentowano bloki kodu uruchamiające trzy ze wspomnianych ekranów. Wykorzystując panel **Bloki**, wybierając dowolny komponent typu przycisk, można znaleźć blok komponentowy z akcją **gdy Nazwa_przycisku.Klik wykonaj**, która pozwala na reakcję na określone zdarzenie – tj. kliknięcie przycisku. Blok wbudowany **otwórz inny ekran** można znaleźć w grupie **Kontrola**, podobnie jak blok **zamknij aplikację**.



Rys. 46. Ekran Screen1 aplikacji Hello_ID w trycie Edytor Blokowy

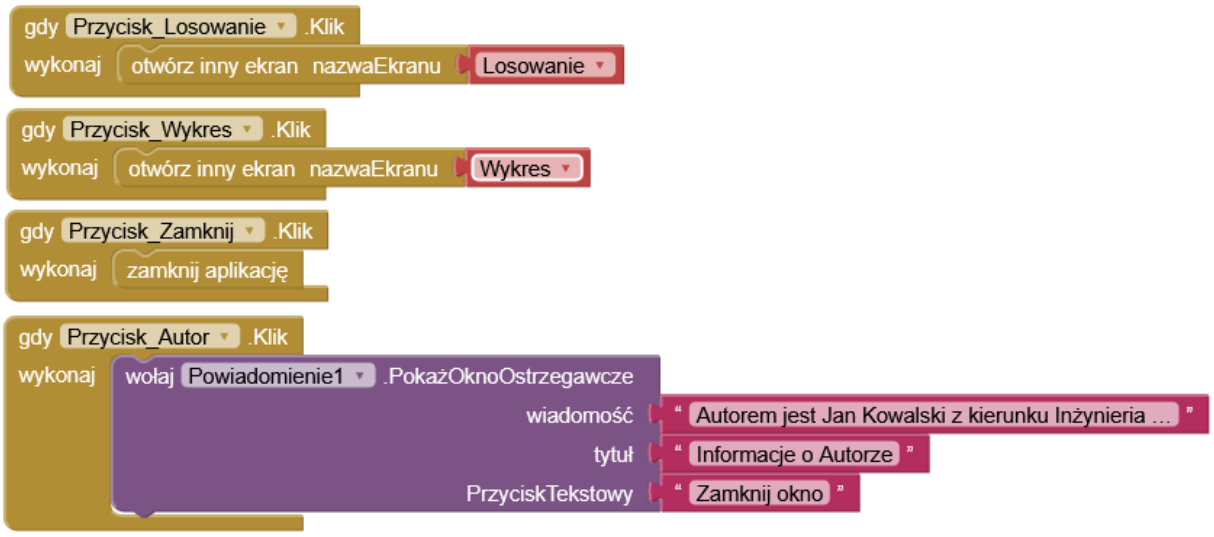
[Tekst alternatywny. Rysunek 46 przedstawia fragment narzędzia MIT App Inventor w trybie Edytor Blokowy. Widoczne są 3 bloki kodu dla ekranu Screen1.]

Jak jednak wyświetlić odpowiedni komunikat po naciśnięciu przycisku **Przycisk_Autor**? Należy ponownie przełączyć się do trybu **Projektant** i dodać komponent niewizualny o nazwie **Powiadomienie** – rys. 47.

[Tekst alternatywny. Rysunek 47 przedstawia dwa fragmenty narzędzia MIT App Inventor w trybie Projektant. Po lewej stronie widoczny jest dół wirtualnego telefonu, pod którym jest komponent niewizualny o nazwie Powiadomienie1, a po prawej stronie – lista komponentów.]

Rys. 47. Nowy komponent – Powiadomienie

Następnie można wrócić z powrotem do trybu **Edytor Blokowy** i dodać ostatni blok kodu, wykorzystując blok komponentowy **wołaj Powiadomienie1.PokażOknoOstrzegawcze**. Całość kodu zaprezentowano poniżej, na rys. 48.



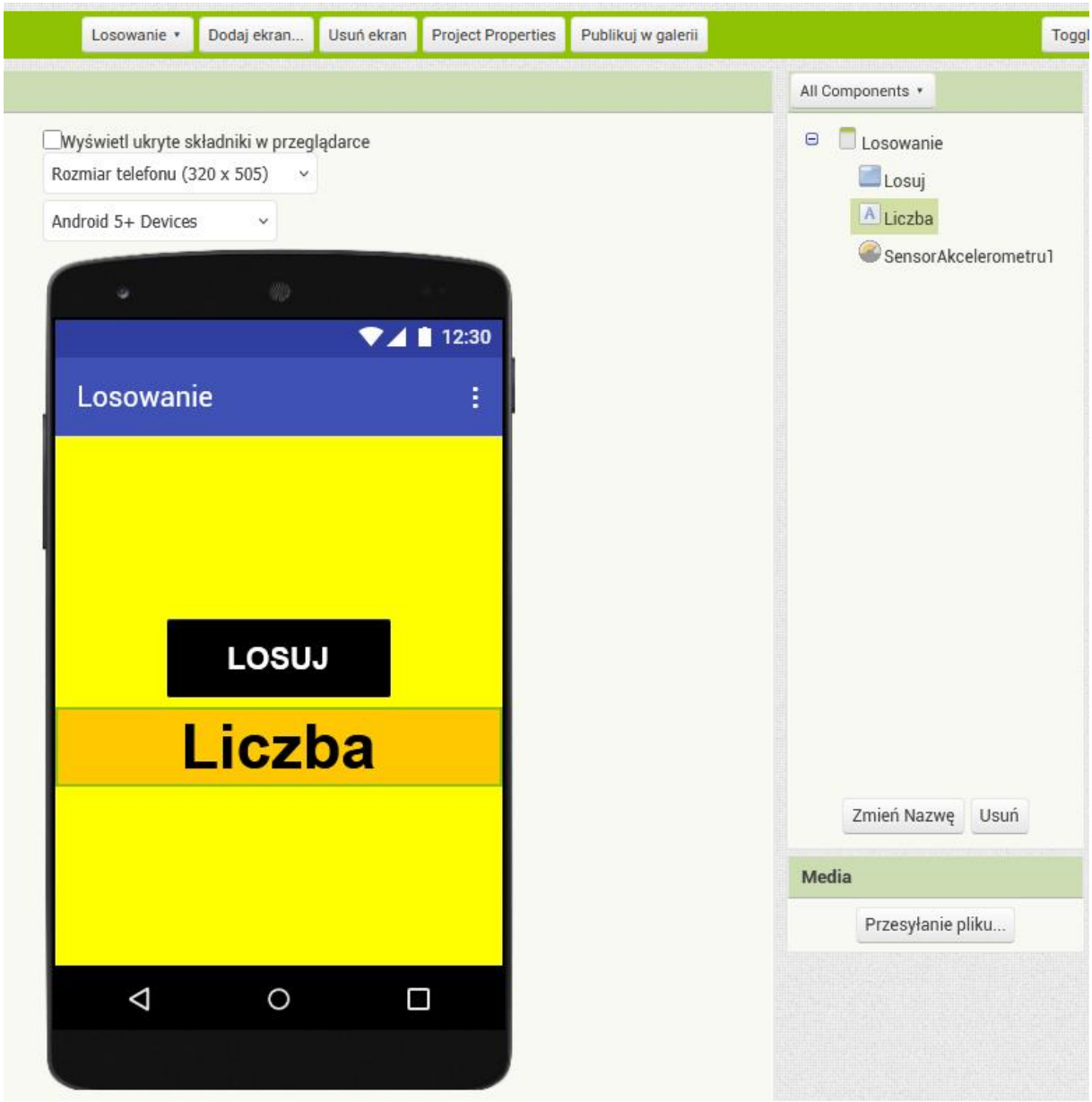
Rys. 48. Bloki kodu dla ekranu Screen1

[Tekst alternatywny. Rysunek 48 przedstawia fragment narzędzia MIT App Inventor w trybie Edytor Blokowy. Widoczne są 4 bloki kodu dla ekranu Screen1, które pozwalają na obsługę przycisków.]

3.3. Ekran Losowanie

Z listy dostępnych ekranów (rys. 45) należy wybrać ten o nazwie **Losowanie**.
Z **Palety**, do obszaru wirtualnego telefonu, należy przeciągnąć 3 komponenty:

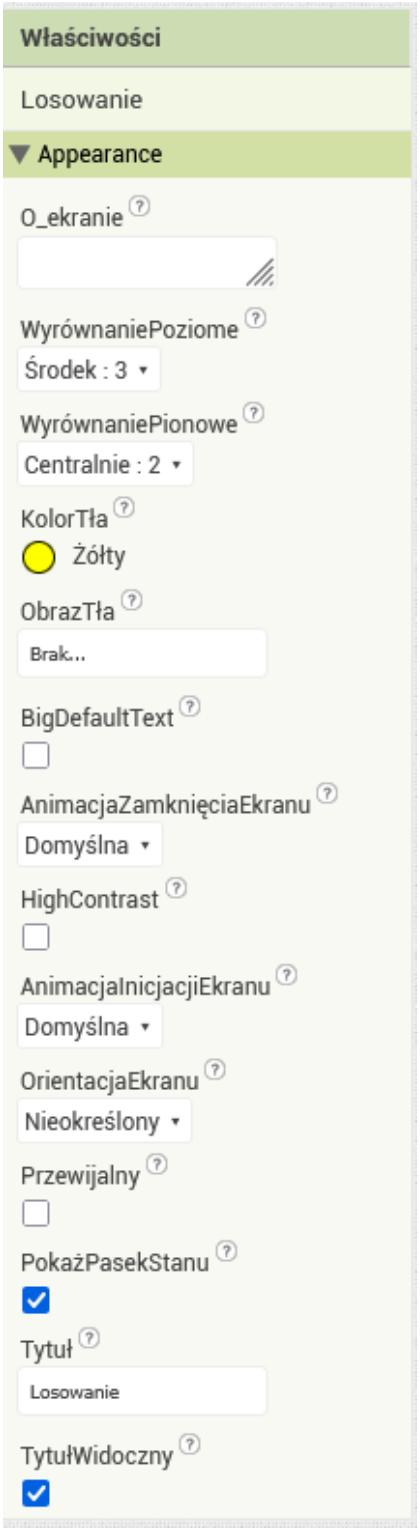
Przycisk (zmieniając jego nazwę na **Losuj**) oraz **Etykieta** (zmieniając jego nazwę na **Liczba**) z palety **Interfejs użytkownika** oraz **SensorAkcelerometru** – komponent niewizualny z grupy **Czujniki**. Ekran ten, po wykonaniu wszystkich poniższych czynności, powinien wyglądać tak, jak na rys. 49.



Rys. 49. Ekran Losowanie

[Tekst alternatywny. Rysunek 49 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant – wirtualny telefon. Zaprezentowany jest oczekiwany wygląd ekranu Losowanie, m.in. żółty kolor tła, przycisk (biały napis LOSUJ na czarnym tle)

oraz etykieta (czarny napis Liczba na pomarańczowym tle). Po prawej stronie widać listę komponentów wraz z ich nazwami.]



Rys. 50. Właściwości dla komponentu Losowanie

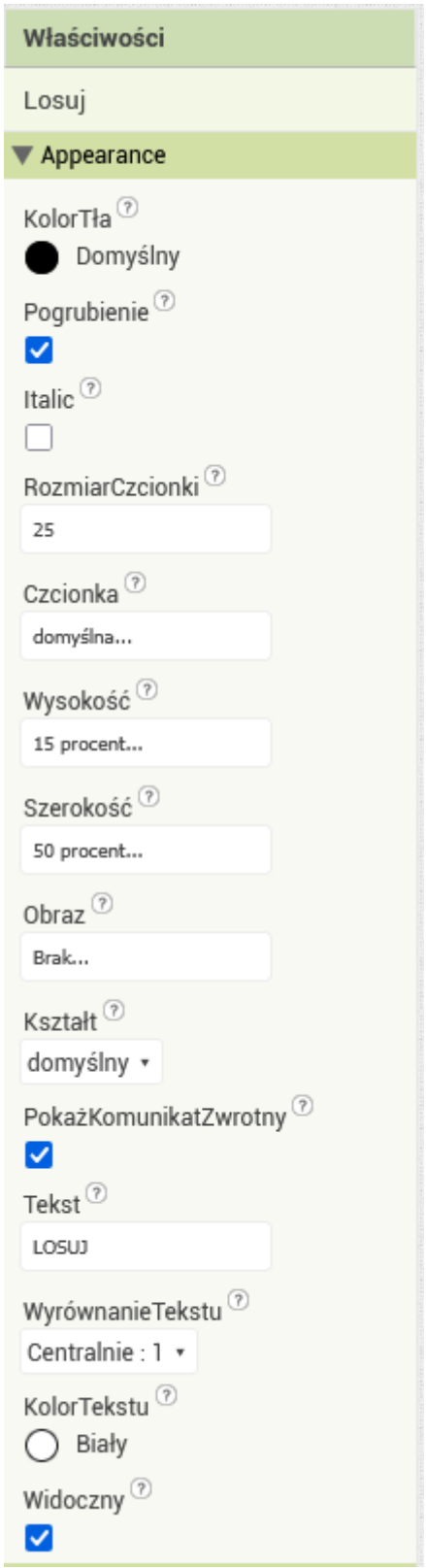
[Tekst alternatywny. Rysunek 50 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu Losowanie.]

W kolejnym kroku należy zmienić wybrane właściwości komponentów:

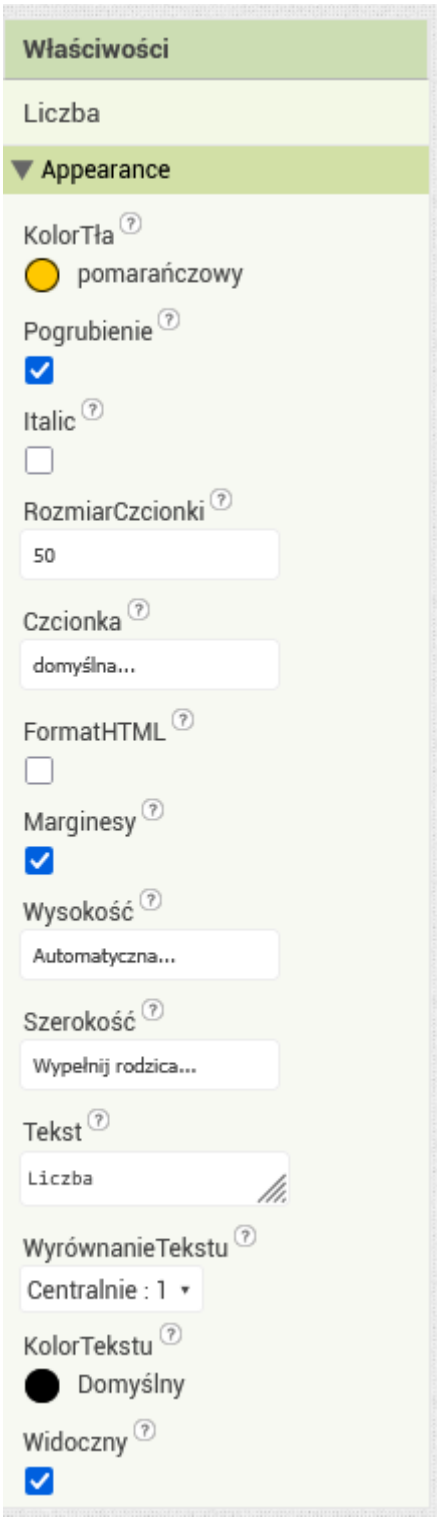
a) Dla komponentu **Losowanie** ustawić (rys. 50; reszta – ustawienia domyślne):
   a. **KolorTła** – **Żółty**,
   b. **WyrównaniePoziome** – **Środek**,
   c. **WyrównaniePionowe** – **Centralnie**.
b) Dla przycisku **Losuj** ustawić (rys. 51; reszta – ustawienia domyślne):
   a. **KolorTła**: **Czarny**,
   b. **Pogrubienie**: **Tak**,
   c. **RozmiarCzcionki**: **25**,
   d. **Wysokość**: **15 procent**,
   e. **Szerokość**: **50 procent**,
   f. **Tekst**: wpisać: **LOSUJ**,
   g. **KolorTekstu**: **Biały**.
c) Dla etykiety **Liczba** ustawić (rys. 52; reszta – ustawienia domyślne):
   a. **KolorTła**: **Pomarańczowy**,
   b. **Pogrubienie**: **Tak**,
   c. **RozmiarCzcionki**: **50**,
   d. **Wysokość**: **Automatyczna**,
   e. **Szerokość**: **Wypełnij rodzica**,
   f. **Tekst**: wpisać: **Liczba**,
   g. **WyrównanieTekstu**: **Centralnie**.
d) Dla komponentu **SensorAkcelerometru1** – ustawienia domyślne.

[Tekst alternatywny. Rysunek 51 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla przycisku Losuj.]

[Tekst alternatywny. Rysunek 52 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla etykiety Liczba.]

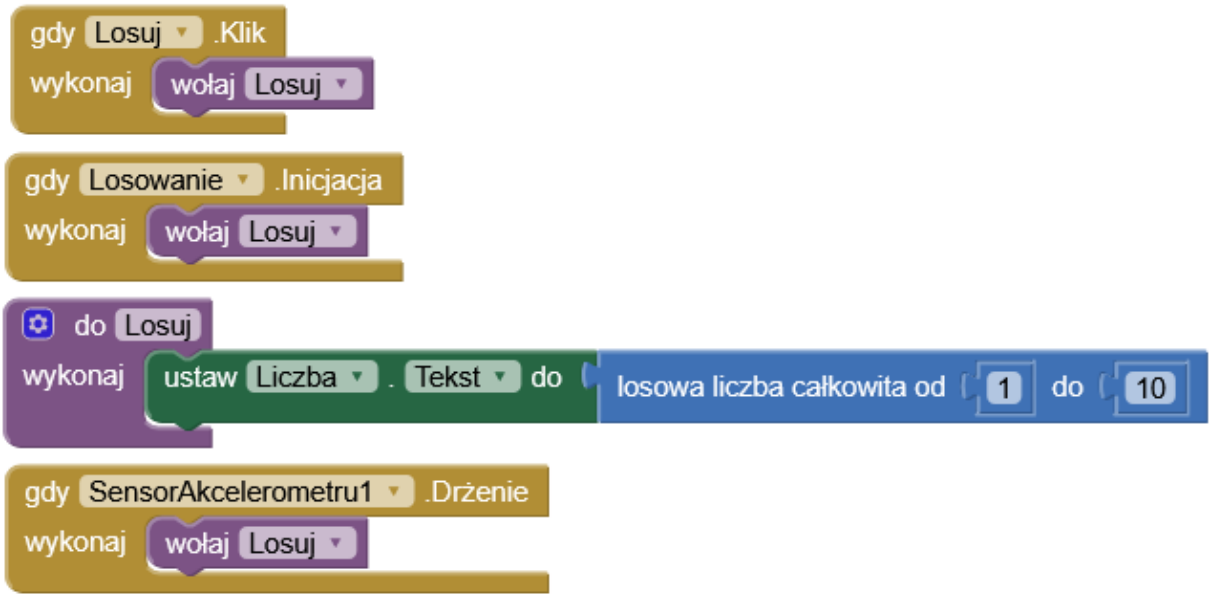


Rys. 51. Właściwości dla komponentu Losuj



Rys. 52. Właściwości dla komponentu Liczba

Po zdefiniowaniu wyglądu ekranu **Losowanie** należy przejść do trybu **Edytor Blokowy** dla tego ekranu i stworzyć odpowiednie bloki kodu – rys. 53. Utworzona została procedura **Losuj**, która pozwala na losowanie liczby całkowitej z zakresu od 1 do 10 (bloki wbudowane z grupy **Matematyczne**), a wynik zwracany jest do etykiety **Liczba** (właściwość **Tekst** – blok komponentowy). Wywołanie tej procedury (blok wbudowany **wołaj** z grupy **Funkcje**) jest podczepione pod 3 inne bloki komponentowe: gdy przycisk **Losuj** zostanie kliknięty, gdy ekran **Losowanie** zostanie zainicjowany oraz gdy zostanie wykryte drżenie (tj. potrząśnięcie telefonem – komponent **SensorAkcelerometru1**).

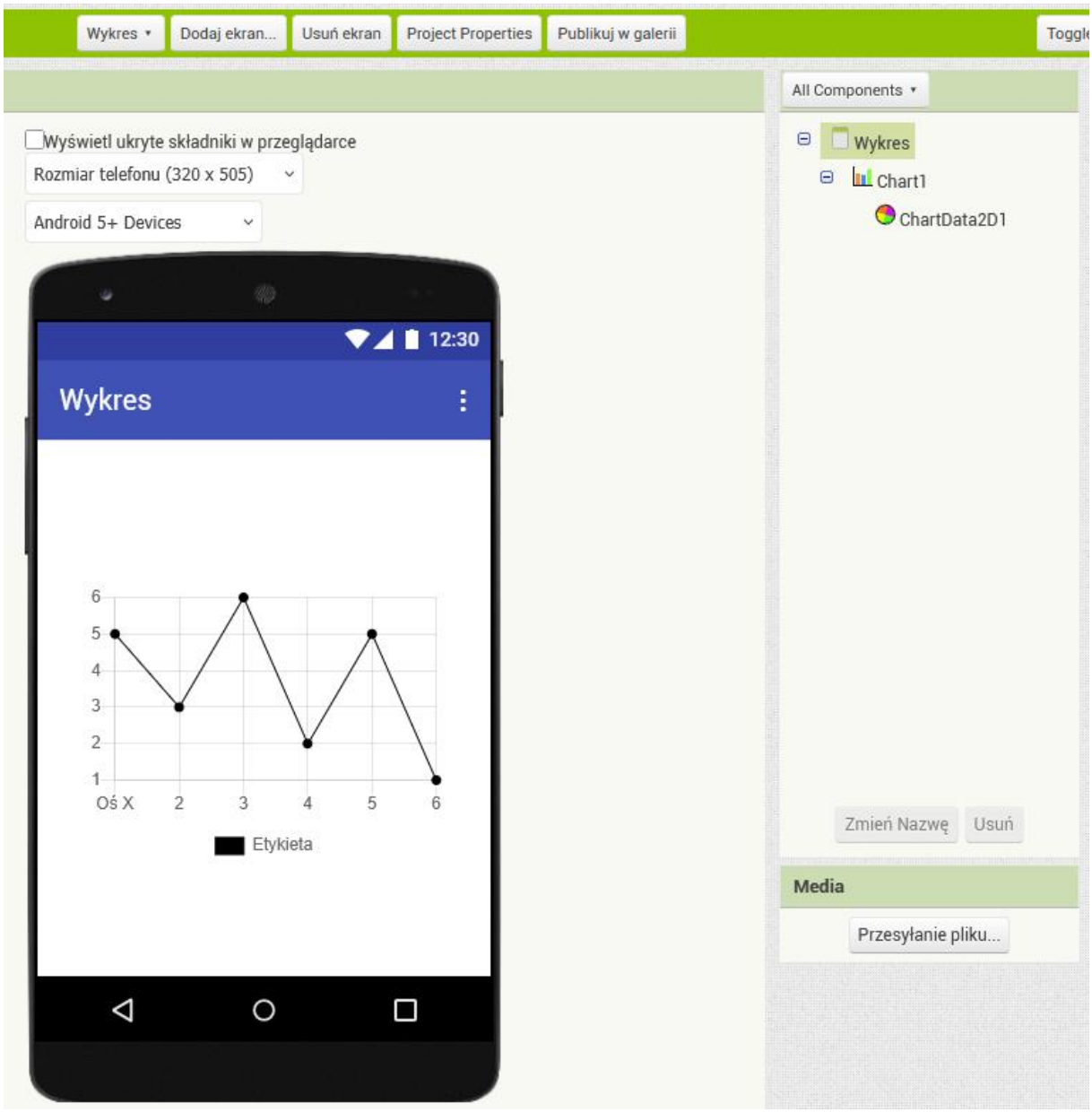


Rys. 53. Bloki kodu dla ekranu Screen1

[Tekst alternatywny. Rysunek 53 przedstawia fragment narzędzia MIT App Inventor w trybie Edytor Blokowy. Widoczne są 4 bloki kodu dla ekranu Losowanie, w tym definicja procedury Losuj, która pozwala na losowanie liczby całkowitej z zakresu od 1 do 10.]

3.4. Ekran Wykres

Z listy dostępnych ekranów (rys. 45) należy wybrać ten o nazwie **Wykres**. Z **Palety**, do obszaru wirtualnego telefonu, należy przeciągnąć 2 komponenty z grupy **Charts**: **Chart**, a następnie **ChartData2D** (należy przeciągnąć go do „wnętrza komponentu **Chart**"). Ekran ten, po wykonaniu wszystkich poniższych czynności, powinien wyglądać tak, jak na rys. 54.



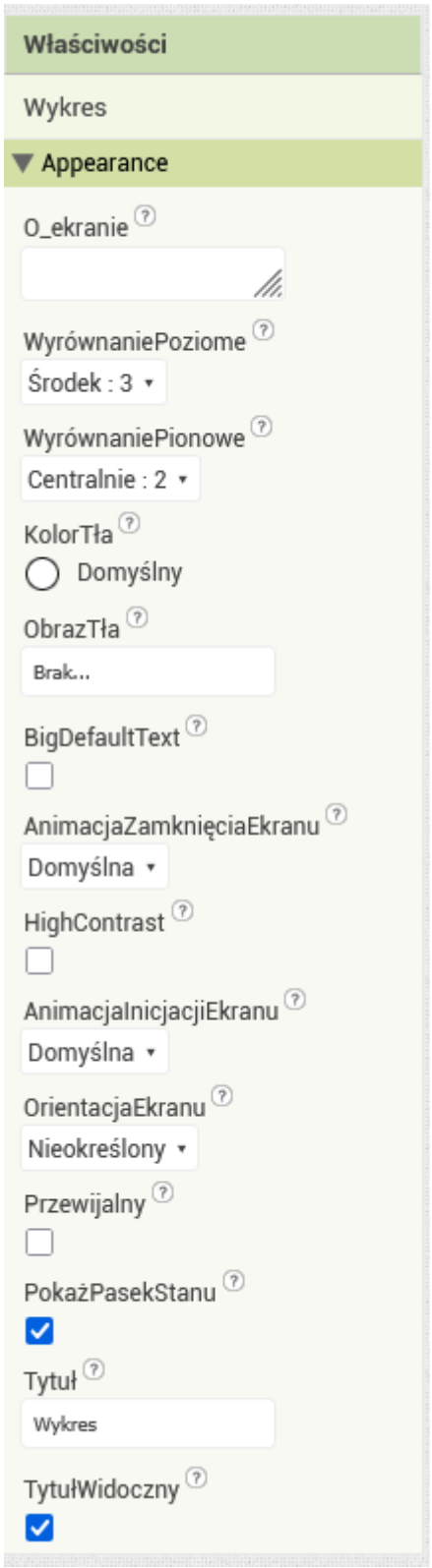
Rys. 54. Ekran Wykres

[Tekst alternatywny. Rysunek 54 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant – wirtualny telefon. Zaprezentowany jest oczekiwany wygląd ekranu Wykres, obejmujący przykładowy wykres liniowy. Po prawej stronie widać listę komponentów wraz z ich nazwami.]

W kolejnym kroku należy zmienić wybrane właściwości komponentów:

a) Dla komponentu **Wykres** (rys. 55; reszta – ustawienia domyślne):
   a. **WyrównaniePoziome** – **Środek**,
   b. **WyrównaniePionowe** – **Centralnie**.

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki” nr FERS.01.05-IP.08-0234/23



Rys. 55. Właściwości dla komponentu Wykres

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*

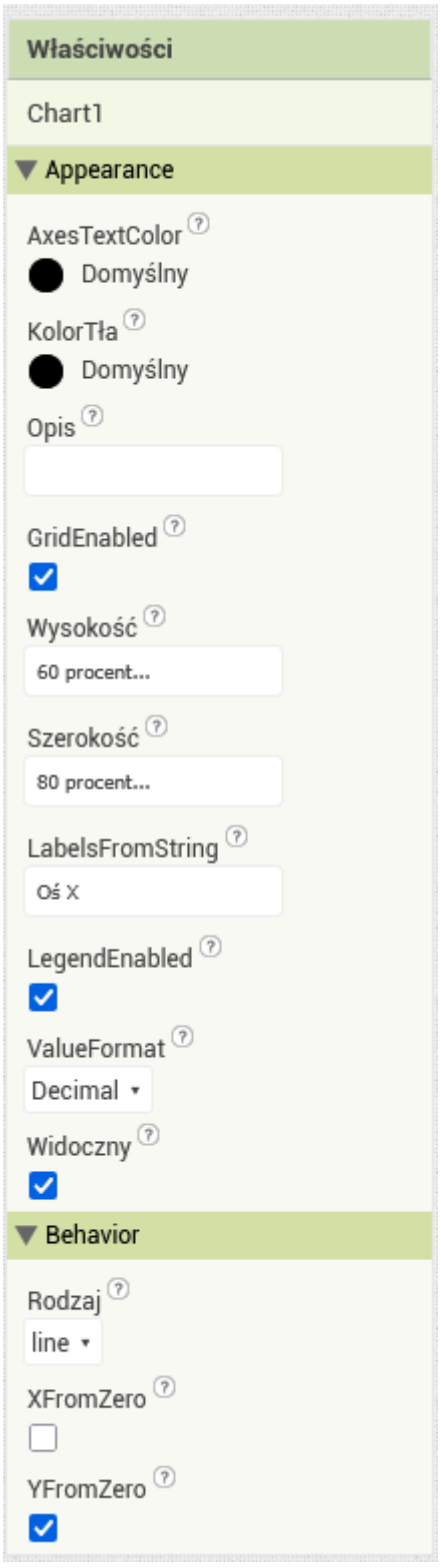
[Tekst alternatywny. Rysunek 55 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu Wykres.]

b) Dla komponentu **Chart1** (rys. 56; reszta – ustawienia domyślne):
   a. **Wysokość**: **60 procent**,
   b. **Szerokość**: **80 procent**,
   c. **LabelsFromString**: Oś X,
   d. **Rodzaj**: **Line**; wykres liniowy,
   e. **YFromZero**: **Tak**; skala na osi Y zaczyna się od 0.

c) Dla komponentu **ChartData2D1** (rys. 57; reszta – ustawienia domyślne):
   a. **Label**: **Etykieta**,
   b. **ElementsFromPairs**: **1,5,2,3,3,6,4,2,5,5,6,1**; wykreślane punkty: x1, y1, x2, y2, … .

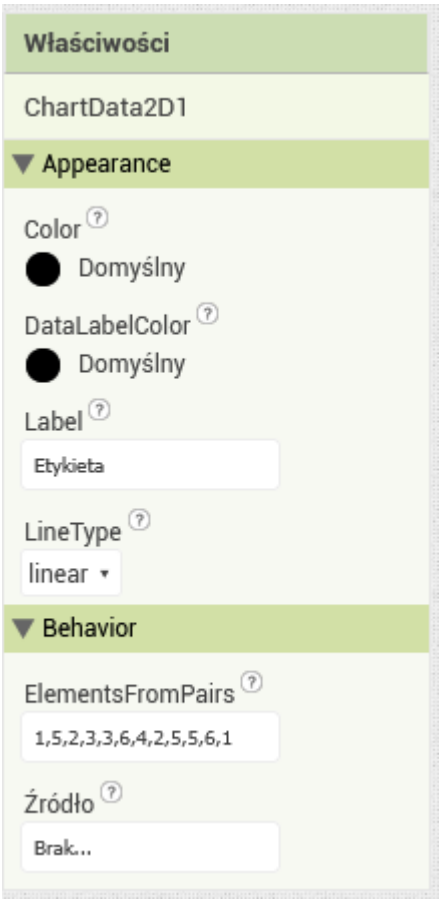
[Tekst alternatywny. Rysunek 56 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu Chart1.]

[Tekst alternatywny. Rysunek 57 przedstawia fragment narzędzia MIT App Inventor w trybie Projektant. Widoczne są właściwości zdefiniowane dla komponentu ChartData2D1.]



Rys. 56. Właściwości dla komponentu Chart1

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23



Rys. 57. Właściwości dla komponentu ChartData2D1

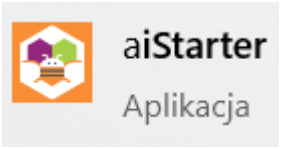
Funkcjonowanie tego ekranu nie wymaga tworzenia żadnych bloków w trybie **Edytor Blokowy**.

3.5. Testowanie aplikacji

Aplikację przetestowano zarówno przy wykorzystaniu emulatora, jak i fizycznego smartfona z systemem mobilnym Android.

3.5.1. Testowanie przy wykorzystaniu emulatora

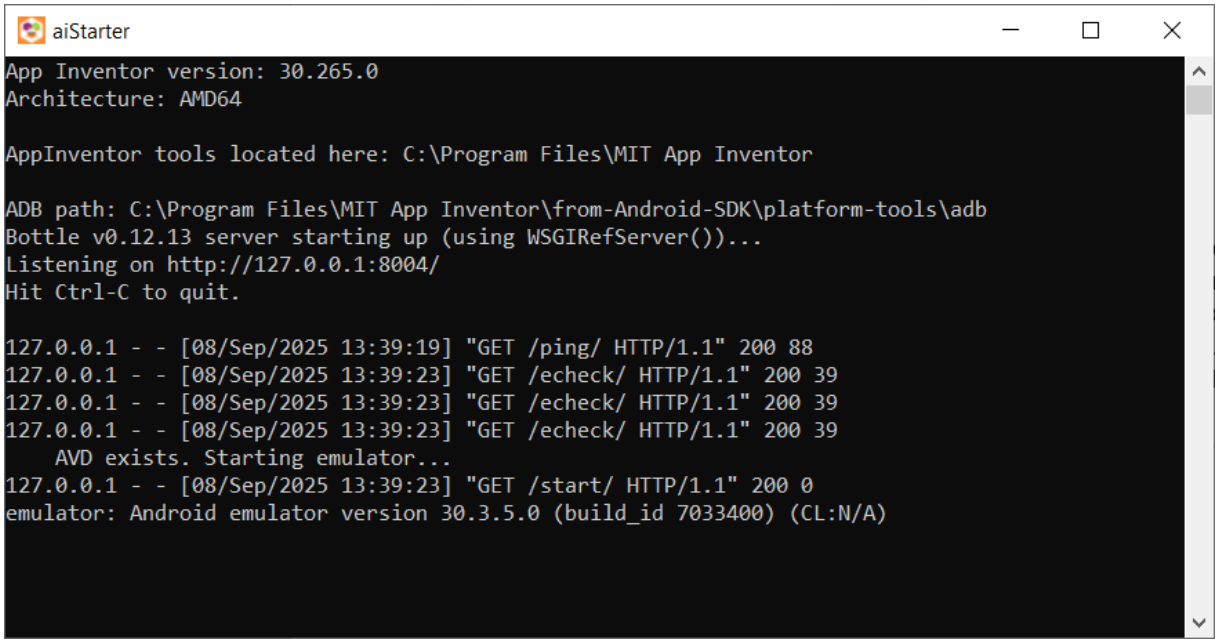
W celu skorzystania z emulatora systemu Android na komputerze z systemem Windows, należy włączyć w tle program **aiStarter** (domyślnie: *C:\Program Files\MIT App Inventor\aiStarter.exe*). Skrót można odnaleźć też w Menu Start (rys. 58).



Rys. 58. Uruchamianie programu aiStarter – ikona widoczna w Menu Start

[Tekst alternatywny. Rysunek 58 przedstawia ikonę programu aiStarter widoczną w Menu Start.]

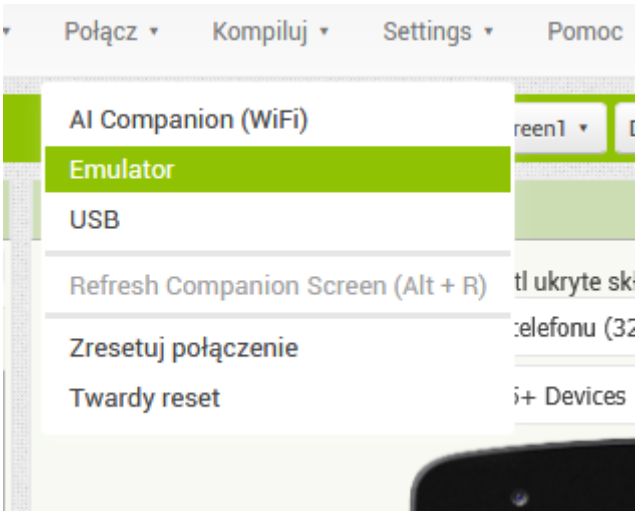
Należy pamiętać, że podczas korzystania z emulatora, program **aiStarter** (rys. 59) musi być bezwzględnie uruchomiony w tle.



Rys. 59. Program aiStarter

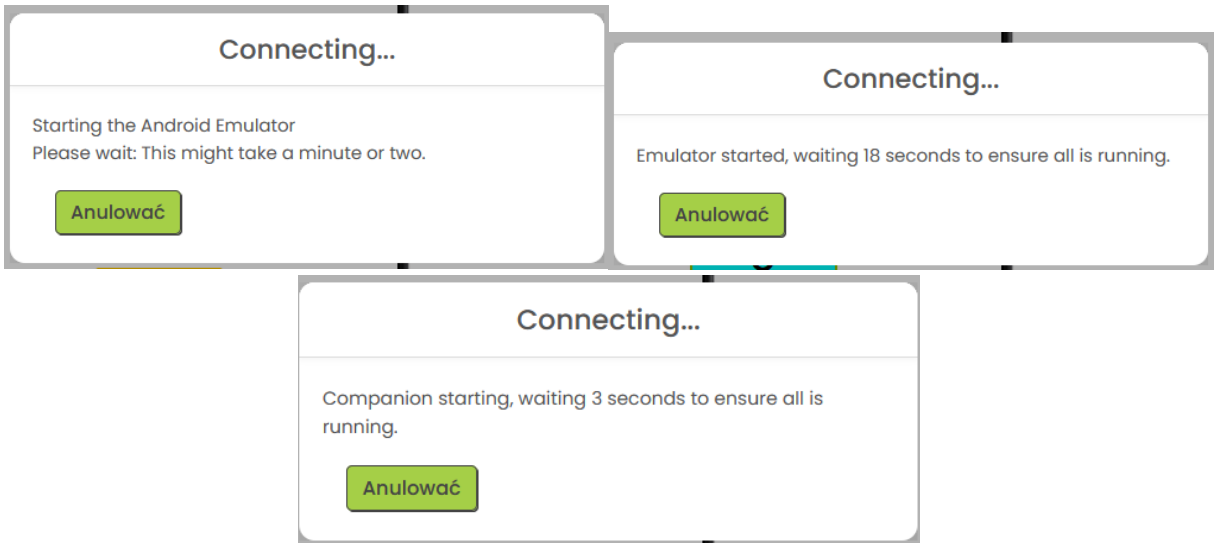
[Tekst alternatywny. Rysunek 59 przedstawia program aiStarter.]

Następnie, z górnego menu nawigacyjnego narzędzia MIT App Inventor należy wybrać opcję **Połącz**, a następnie **Emulator** – rys. 60. Trzeba postępować zgodnie z komunikatami wyświetlanymi na ekranie – a jest ich dużo (rys. 61). Uwaga: emulator potrzebuje dłuższą chwilę czasu, aby się włączyć, szczególnie przy pierwszym uruchomieniu. W celu bezproblemowej pracy należy także upewnić się, czy zainstalowana jest na nim aktualna wersja oprogramowania **MIT AI2 Companion**. W przypadku problemów z połączeniem należy wykorzystać opcję **Zresetuj połączenie**. Należy nie klikać na opcję **Twardy reset**, gdyż przywraca ona emulator do ustawień domyślnych!



Rys. 60. Łączenie MIT App Inventora z emulatorem

[Tekst alternatywny. Rysunek 60 przedstawia fragment narzędzia MIT App Inventor – zakładkę Połącz w górnym menu nawigacyjnych. Opcja Emulator pozwala na połączenie z emulatorem.]



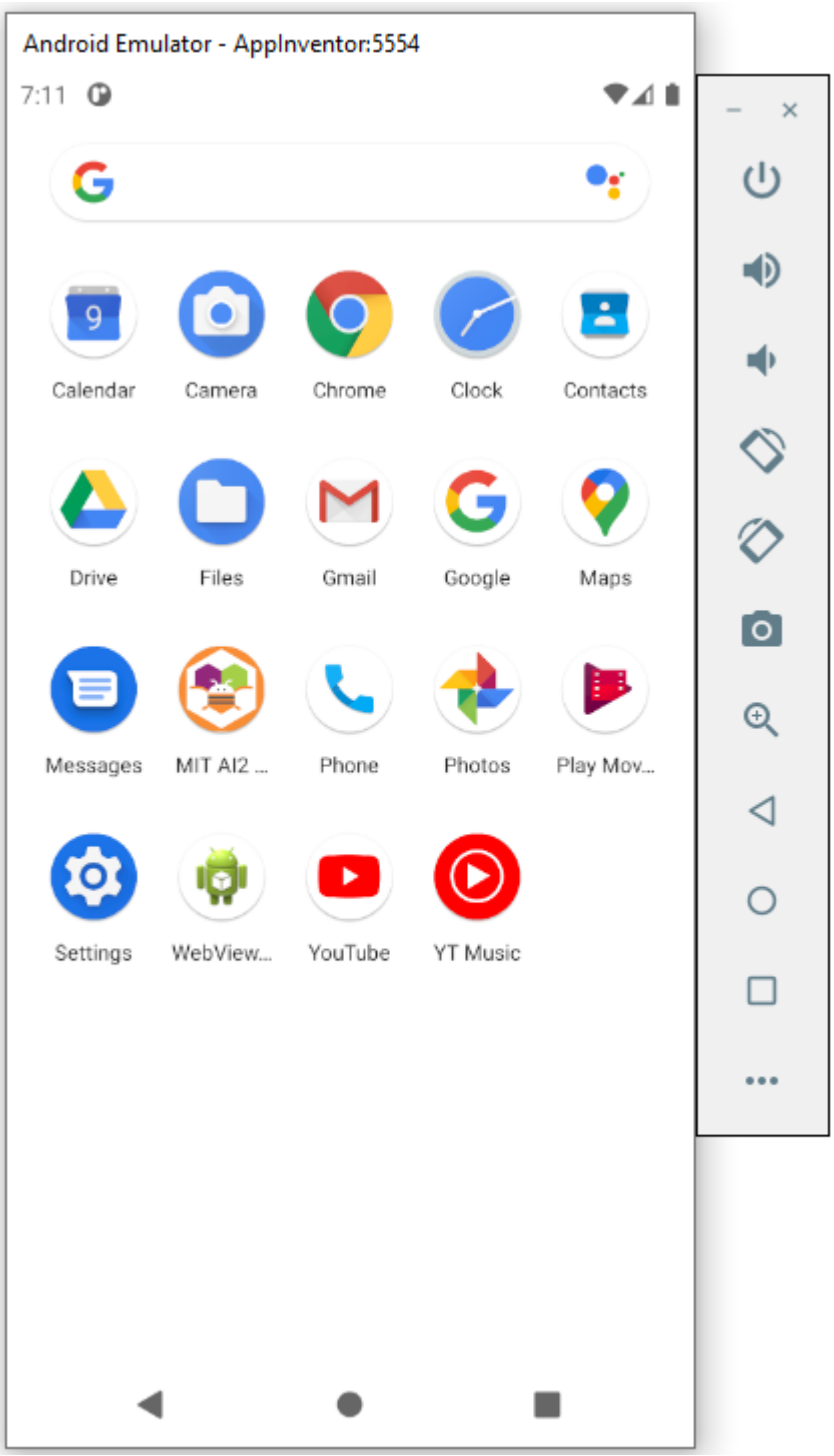
Rys. 61. Przykładowe informacje pojawiające się w trakcie łączenie MIT App Inventora z emulatorem

[Tekst alternatywny. Rysunek 61 przedstawia 3 fragmenty okien w narzędziu MIT App Inventor – przykładowe komunikaty pojawiające się w trakcie łączenie MIT App Inventora z emulatorem.]

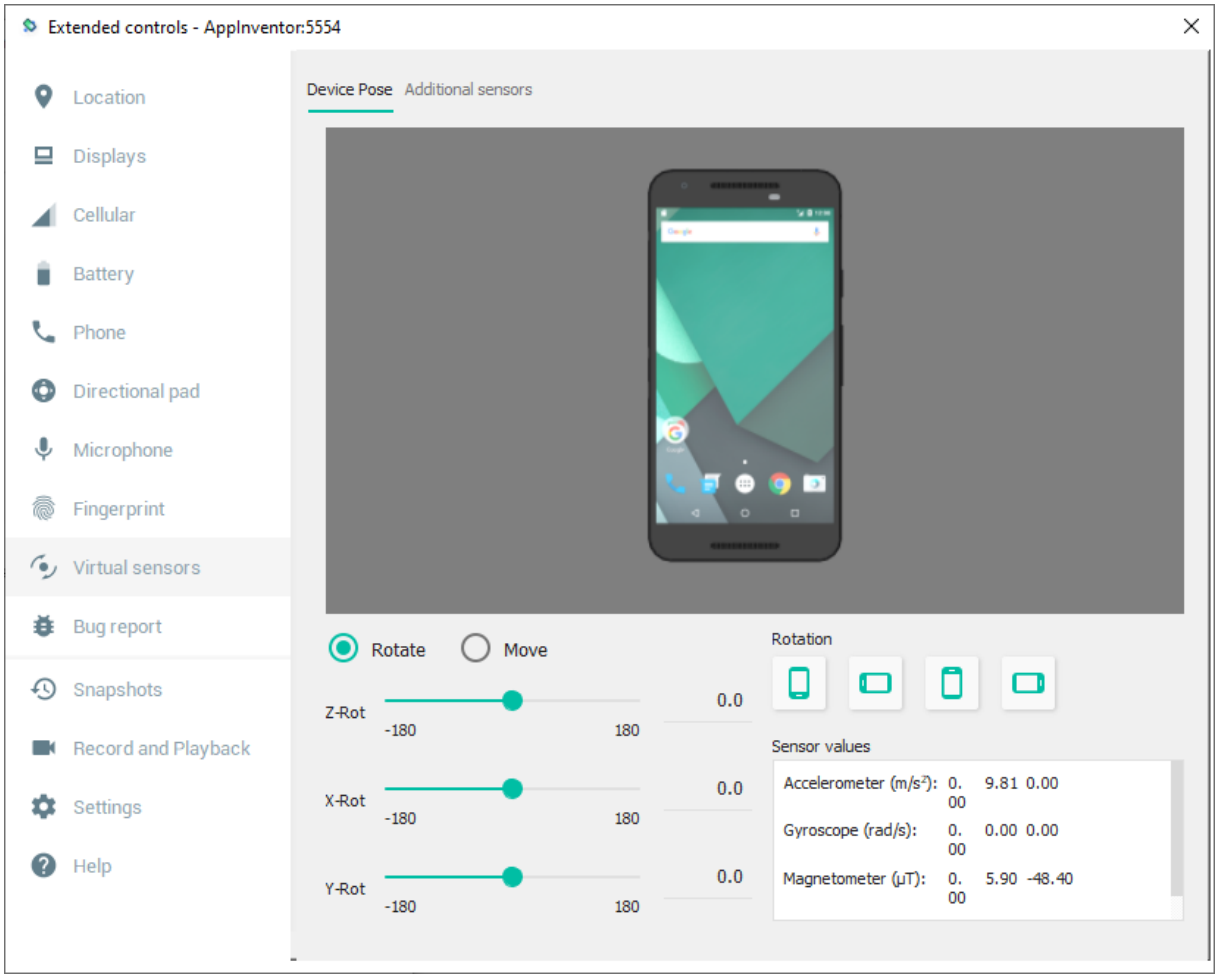
Po chwili emulator powinien się uruchomić i będzie działał jak wirtualny smartfon z systemem Android, który jest uruchamiany bezpośrednio na komputerze (rys. 62) – przez co pozwala testować aplikacje bez fizycznego urządzenia mobilnego. Na liście aplikacji znajduje się m.in. **MIT AI2 Companion** – specjalny program, który umożliwia testowanie projektów stworzonych w narzędziu MIT App Inventor w czasie rzeczywistym; po sparowaniu go z projektem w MIT App Inventor, od razu uruchamiana jest tworzona aplikacja mobilna – bez konieczności wcześniejszego kompilowania i instalowania pliku ***.apk**. Dzięki temu programista od razu widzi efekty swojej pracy. Emulator ma dostęp do wszystkich standardowych funkcji systemu mobilnego Android. Po prawej stronie ekranu emulatora znajduje się **pasek narzędzi**, który stanowi swoisty symulator przycisków i opcji urządzenia – pozwala na interakcję z wirtualnym urządzeniem. Znajdują się tam przyciski odpowiadające fizycznym opcjom urządzenia, np. *power*, *cofnij*, *ekran główny*, *regulacja głośności*, *zrzut ekranu*, *symulacja obrotu urządzenia*. Na samym dole znajduje się także przycisk z trzema kropkami, który uruchamia zaawansowany panel (rys. 63) pozwalający na symulację i kontrolę wielu funkcji wirtualnego urządzenia; jest to często niezbędne do testowania aplikacji korzystających z nietypowych sensorów i opcji, np. zmiana lokalizacji GPS, zmiana rozdzielczości i gęstości pikseli ekranu, zmiana statusu połączenia komórkowego i stanu baterii, czy też symulacja: połączeń telefonicznych, wiadomości SMS, mikrofonu, czytnika linii papilarnych i różnorodnych czujników urządzenia. Dzięki dostępnym opcjom, możliwe jest wierne odtworzenie warunków pracy rzeczywistego urządzenia i przetestowanie tworzonej aplikacji w różnych scenariuszach.

[Tekst alternatywny. Rysunek 62 przedstawia wirtualny smartfon z systemem mobilnym Android (tj. emulator systemu Android), uruchomiony w kontekście pracy z narzędziem MIT App Inventor. Na ekranie wirtualnego urządzenia widać listę dostępnych aplikacji (w tym MIT AI2 Companion), a po prawej stronie znajduje się pasek narzędzi, który stanowi swoisty symulator przycisków i opcji urządzenia.]

[Tekst alternatywny. Rysunek 63 przedstawia okno Extended controls – panel emulatora systemu Android w narzędziu MIT App Inventor, gdzie znajdują się opcje umożliwiające symulację różnych funkcji urządzenia mobilnego, np.: lokalizacja GPS, poziom baterii, połączenie sieciowe, odczyty czujników, połączenia telefoniczne, wiadomości SMS, kamera, mikrofon, czytnik linii papilarnych, zmiana rozdzielczości i gęstości pikseli ekranu.]

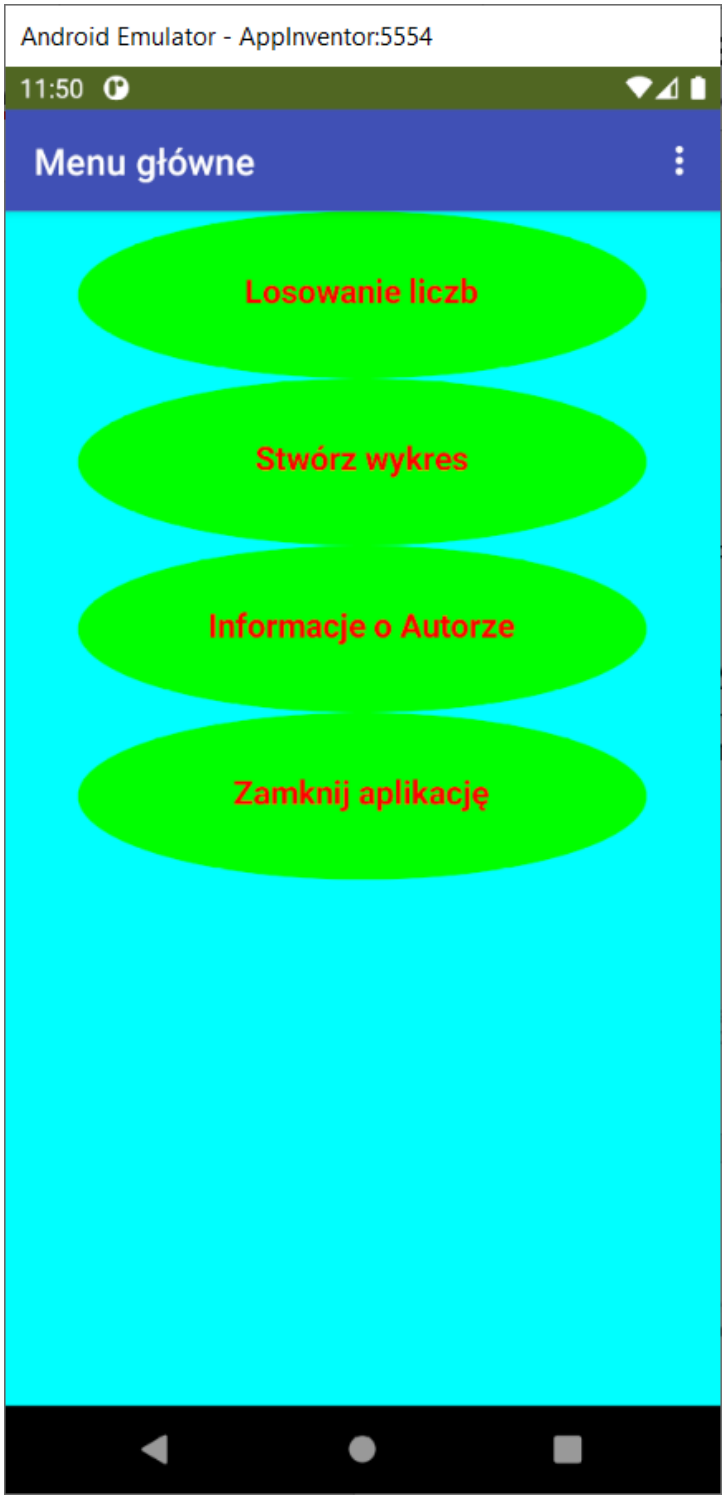


Rys. 62. Emulator systemu Android dla narzędzia MIT App Inventor



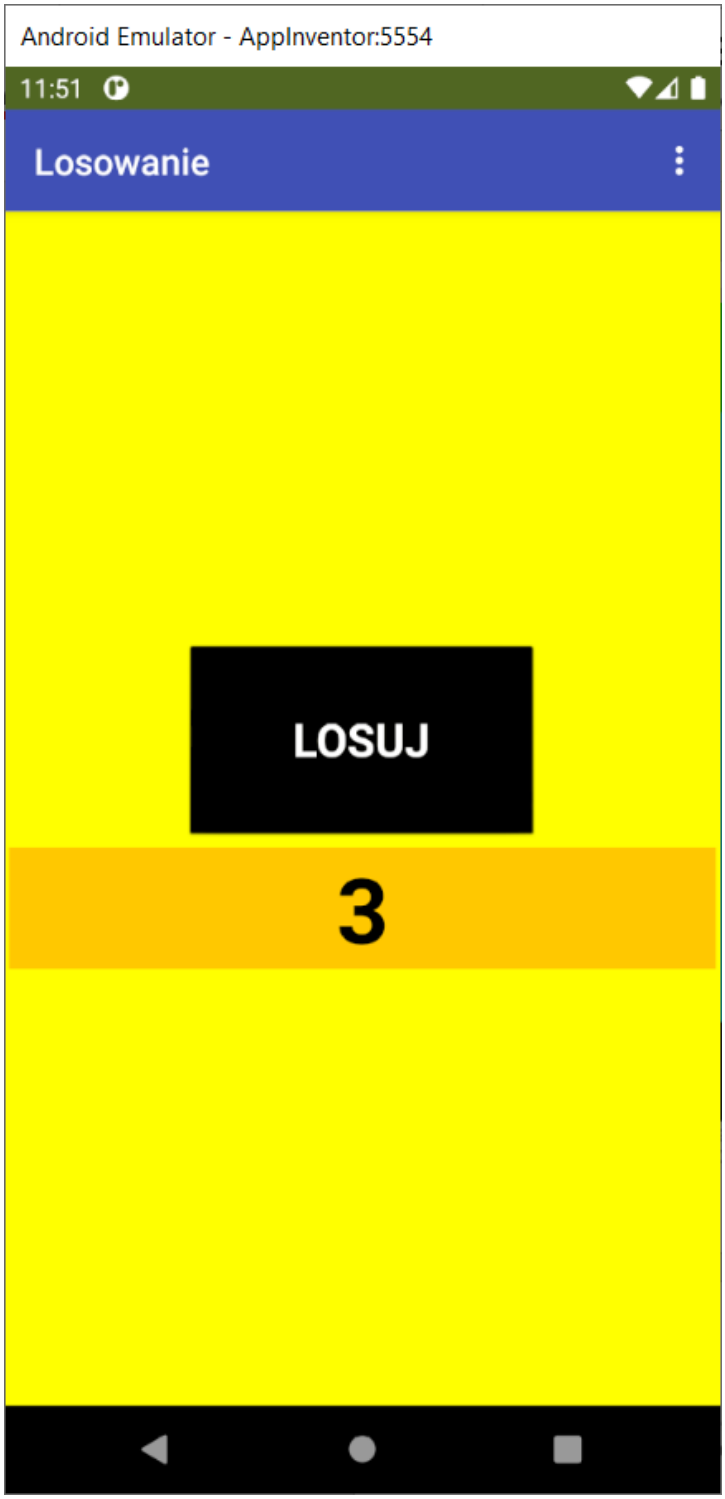
Rys. 63. Okno z ustawieniami emulatora systemu Android w narzędziu MIT App Inventor

Poniżej zaprezentowano wygląd menu głównego (rys. 64), ekranu **Losowanie** (rys. 65) i komunikatu z informacjami o Autorze (po naciśnięciu przycisku Informacje o Autorze w menu głównym – rys. 66) w emulatorze systemu Android.



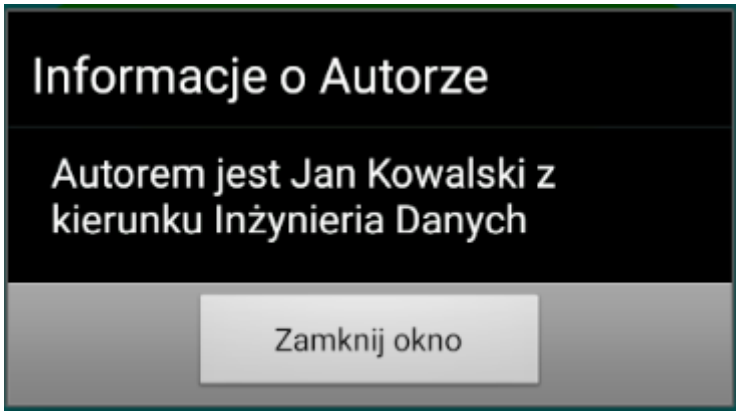
Rys. 64. Menu główne

[Tekst alternatywny. Rysunek 64 przedstawia wygląd menu głównego (ekran Screen1) aplikacji Hello_ID w emulatorze systemu Android.]



Rys. 65. Ekran Losowanie

[Tekst alternatywny. Rysunek 65 przedstawia wygląd ekranu Losowanie aplikacji Hello_ID w emulatorze systemu Android.]



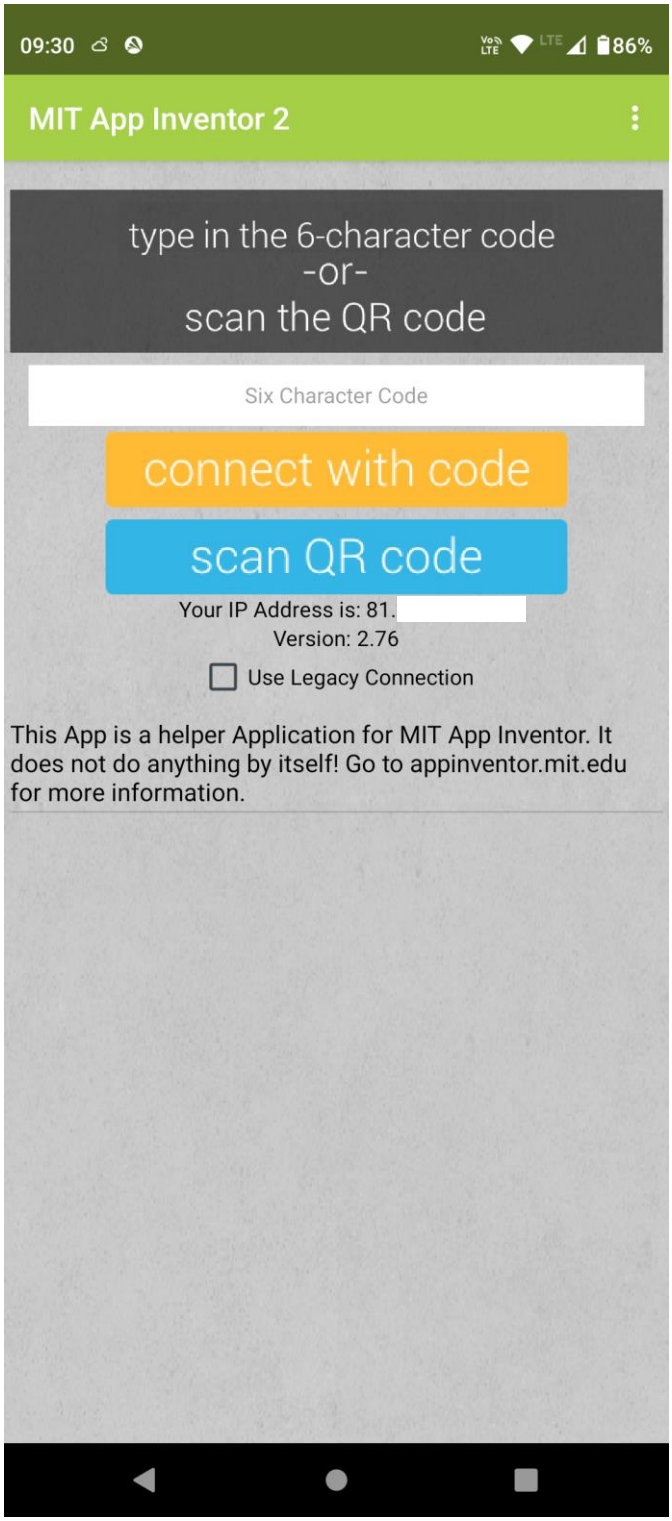
Rys. 66. Komunikat z informacjami o Autorze

[Tekst alternatywny. Rysunek 66 przedstawia wygląd komunikatu z informacjami o Autorze, po naciśnięciu przycisku Informacje o Autorze w menu głównym. Tekst: „Autorem jest Jan Kowalski z kierunku Inżynieria Danych".]

3.5.2. Testowanie na fizycznym urządzeniu

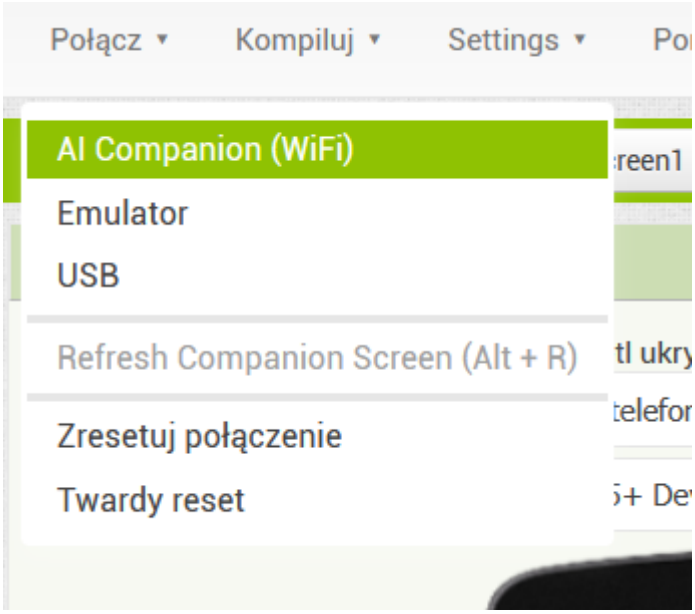
Testowanie aplikacji w czasie rzeczywistym za pomocą fizycznego urządzenia mobilnego (tu: smartfon z systemem Android) wymaga zainstalowania oprogramowaniem **MIT AI2 Companion** (wspomnianego już w podrozdziale 4.4.1.) ze Sklepu Play oraz dostępu do sieci Internet. Po instalacji, program ten należy uruchomić – rys. 67. W przypadku urządzenia fizycznego, należy jednak samodzielnie nawiązać połączenie. Centralna część ekranu **MIT AI2 Companion** zawiera krótką instrukcję dla użytkownika: *wpisz 6-znakowy kod -lub- zeskanuj kod QR*. Są to dwie główne metody pozwalające na połączenie się z projektem tworzonym w narzędziu MIT App Inventor na komputerze – por. poniżej.

[Tekst alternatywny. Rysunek 67 przedstawia screen ekranu z fizycznego urządzenia z systemem mobilnym Android, obejmujący program MIT AI2 Companion, który pozwala na podgląd i interakcję z tworzoną aplikacją w czasie rzeczywistym. Widoczne są opcje: „connect with code" oraz „scan QR code".]



Rys. 67. Program MIT AI2 Companion uruchomiony na fizycznym urządzeniu z systemem Android

Aby połączyć się z fizycznym urządzeniem, w MIT App Inventor należy w górnym menu nawigacyjnym wybrać opcję **Połącz**, a następnie **AI Companion (WiFi)** – rys. 68. Oba urządzenia muszą być podłączone do sieci Internet!

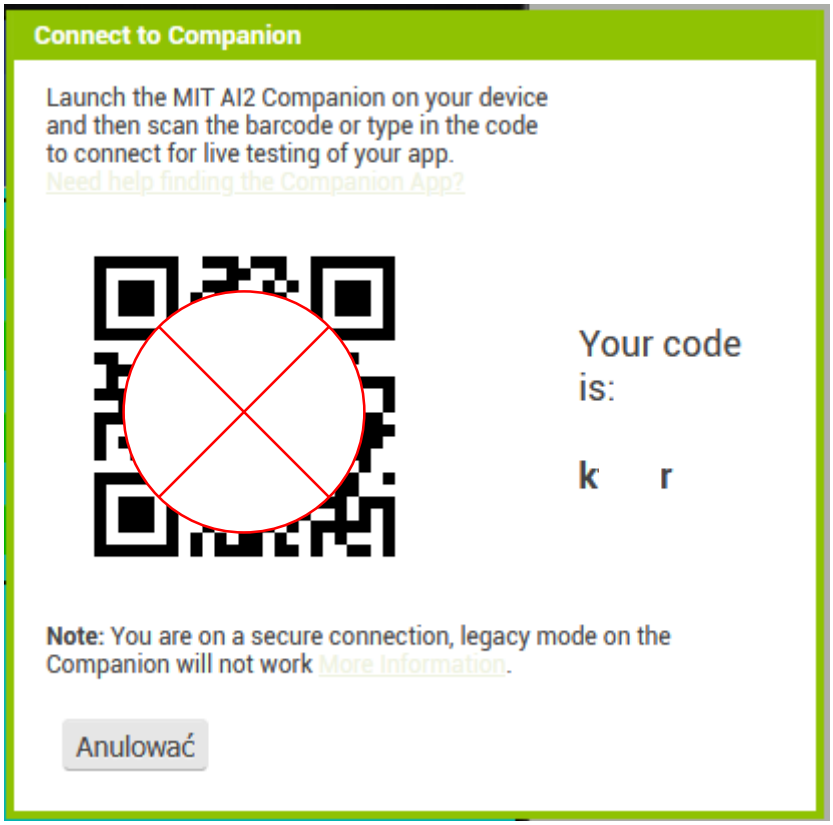


Rys. 68. Łączenie MIT App Inventora z fizycznym urządzeniem

[Tekst alternatywny. Rysunek 68 przedstawia fragment narzędzia MIT App Inventor – zakładkę Połącz w górnym menu nawigacyjnych. Opcja AI Companion (WiFi) pozwala na połączenie z fizycznym urządzeniem poprzez sieć Internet.]

Po chwili powinno pojawić się okno **Connect to Companion** – rys. 69, wraz z widocznym kodem QR oraz sześciocyfrowym kodem. W aplikacji **MIT AI2 Companion** na fizycznym urządzeniu (rys. 67) należy zeskanować ten kod QR lub przepisać go ręcznie do właściwego pola, aby uzyskać połączenie i rozpocząć testowanie aplikacji. Studenci mogą więc naprawdę szybko rozpocząć testowanie swojej aplikacji na własnym urządzeniu mobilnym.

[Tekst alternatywny. Rysunek 69 przedstawia okno „Connect to Companion” z narzędzia MIT App Inventor, które wyświetla kod QR oraz unikalny, sześciocyfrowy kod, pozwalające na połączenie z fizycznym urządzeniem poprzez sieć Internet.]

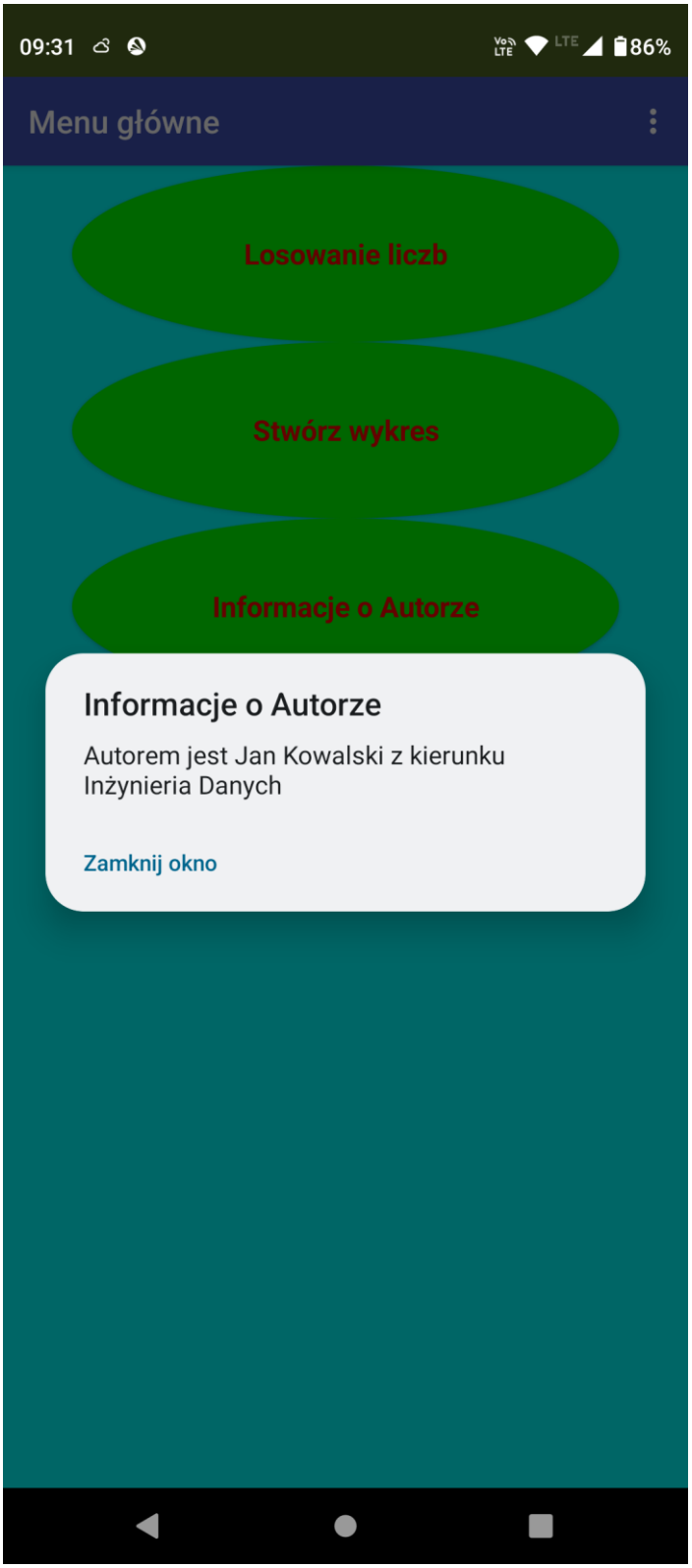


Rys. 69. MIT App Inventor – okno Connect to Companion

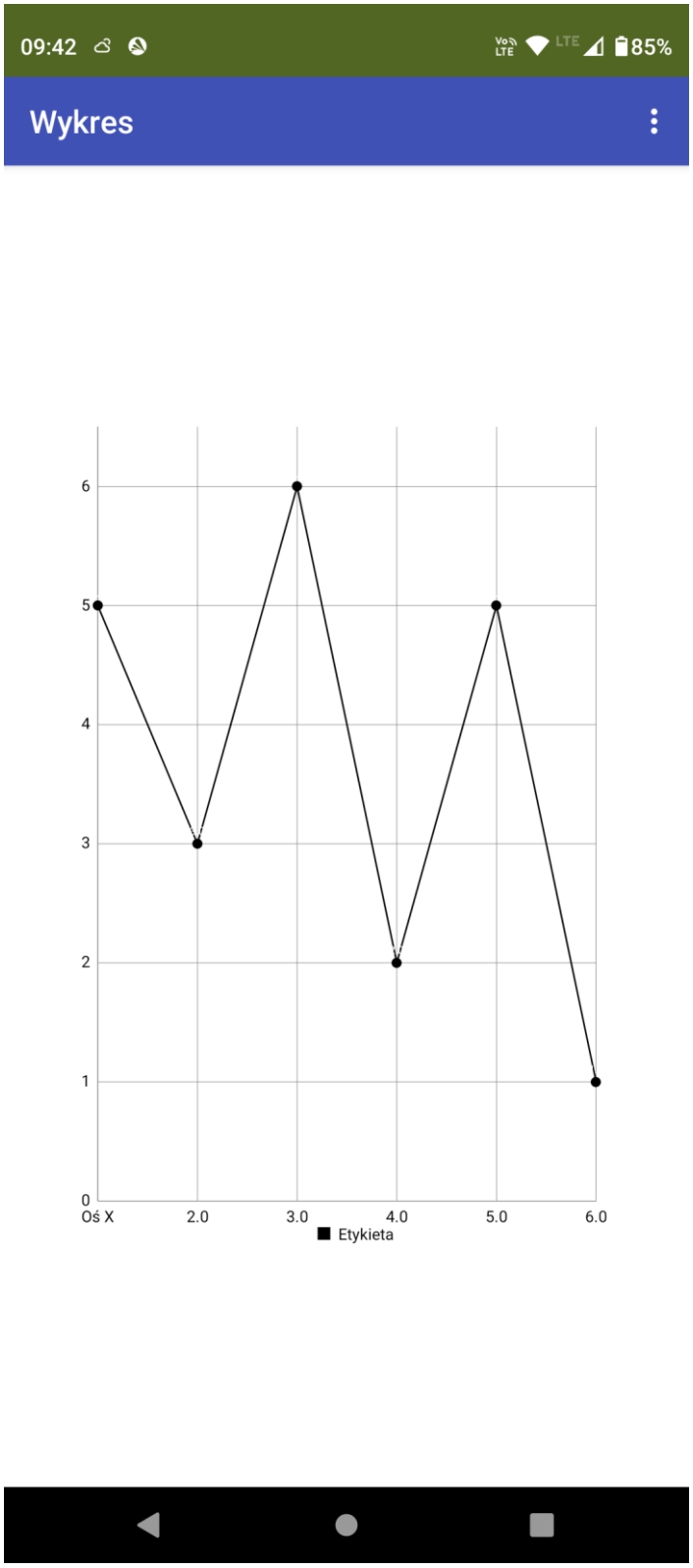
Poniżej zaprezentowano wygląd menu głównego wraz z komunikatem z informacjami o Autorze (rys. 70) oraz ekran **Wykres** (rys. 71) na fizycznym urządzeniu (smartfonie) z systemem operacyjnym Android.

[Tekst alternatywny. Rysunek 70 przedstawia wygląd menu głównego (ekran Screen1) wraz z komunikatem z informacjami o Autorze w aplikacji Hello_ID na fizycznym urządzeniu z systemem Android.]

[Tekst alternatywny. Rysunek 71 przedstawia wygląd ekranu Wykres w aplikacji Hello_ID na fizycznym urządzeniu z systemem Android.]



Rys. 70. Menu główne i komunikat z informacjami o Autorze



Rys. 71. Ekran Wykres

3.6. Zadania do samodzielnego rozwiązania

1. **Ulepsz wygląd aplikacji Hello_ID** – zmień kolory tła, czcionek i kształty przycisków tak, aby aplikacja była bardziej estetyczna.
2. **Nawigacja *wstecz*** – dodaj na ekranach **Losowanie** i **Wykres** przyciski „Powrót”, które przeniosą Użytkownika z powrotem do Menu głównego.
3. **Losowanie większego zakresu** – zmodyfikuj na ekranie **Losowanie** procedurę losowania tak, aby zakres (np. od 1 do 100) można było ustawić ręcznie w aplikacji przy wykorzystaniu odpowiednich komponentów. Ogranicz możliwość wprowadzenia błędnych danych.
4. **Informacje o Autorze** – zamiast komunikatu typu ***MessageDialog***, utwórz dodatkowy ekran **O_autorze**, gdzie zostanie dodane zdjęcie, krótka informacja o Autorze i przycisk powrotu.
5. **Rozbudowa ekranu Losowanie** – dodaj opcję wibracji telefonu po wylosowaniu liczby.
6. **Rozbudowa ekranu Wykres** – zmień dane na wykresie tak, aby były generowane dynamicznie (np. na podstawie losowania lub wpisane przez użytkownika), bądź też wczytane z jakiegoś źródła. Następnie spersonalizuj wygląd tego wykresu według własnego uznania.
7. **Zapis danych** – w ekranie **Losowanie**, dodaj możliwość zapisywania wyników losowania w pamięci trwałej urządzenia i ich późniejszego wyświetlenia na ekranie w formie historii.

Zakończenie

W niniejszych materiałach dydaktycznych do przedmiotu ***Programowanie urządzeń mobilnych*** zwrócono uwagę na powszechność urządzeń mobilnych (np. telefonów komórkowych i tabletów) oraz ich niezwykle istotną rolę w życiu codziennym. Zaakcentowano także, że znajomość narzędzi **no-code** / **low-code** stanowi w dzisiejszych czasach wartość dodaną na rynku pracy i to właśnie tej tematyce poświęcono większość miejsca, co pozwala Studentom na poszerzenie swoich kompetencji w tym zakresie.

Dość szczegółowo omówiono i zaprezentowano możliwość tworzenia aplikacji mobilnych przy wykorzystaniu wybranego narzędzia typu **no-code** / **low-code**: **MIT App Inventor**. Dzięki temu Studenci kierunku ***inżynieria danych*** mogą zacząć tworzyć swoje własne aplikacje mobilne. Niniejsze opracowanie stanowi zarówno poszerzenie treści podstawowych realizowanych w ramach kursu ***Programowanie urządzeń mobilnych*** oraz jednocześnie rozszerza kompetencje Studenta, obejmując zagadnienia wykraczające poza ramy tego kursu.

Student, mając wiedzę na temat jednego narzędzia typu **no-code** / **low-code**, jest w stanie w prosty sposób wykorzystywać także inne narzędzia tego typu. Mając natomiast bazową wiedzę na temat programowania urządzeń mobilnych, może on stopniowo poszerzać swoje umiejętności i tworzyć coraz bardziej wymagające projekty.

Bibliografia

[1]. Android (operating system). (2025-07-13). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=Android_(operating_system)&oldid=1300308672, dostępny 15.07.2025).
[2]. Android Developers. (2025). *Android Mobile App Developer Tools*, (https://developer.android.com, dostępny 10.07.2025).
[3]. App Inventor. (2024). W: *Mistrzowie kodowania*, (https://web.archive.org/web/20250120233146/http://wiki.mistrzowiekodowania.pl/index.php?title=Strona_g%C5%82%C3%B3wna#App_Inventor, dostępny 15.07.2025).
[4]. Apple Developer. (2025). *Develop for iOS*, (https://developer.apple.com/ios/, dostępny 10.07.2025).
[5]. Griffiths D., & Griffiths D. (2018). *Android Programowanie aplikacji. Rusz głową!*, Wydanie II, Helion, Gliwice.
[6]. iOS. (2025-07-11). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=IOS&oldid=129990847, dostępny 15.07.2025).
[7]. Kostereva K., & Kawasaki B. (2022). *The No-Code Playbook*, Creatio, (https://www.creatio.com/sites/default/files/static-site/The-No-Code-Playbook.pdf, dostępny 04.07.2025).
[8]. Krajowy Instytut Mediów. (2025). *Usługi medialne i infrastruktura do ich odbioru w roku 2024. Gospodarstwa domowe*, Warszawa, (https://kim.gov.pl/wp-content/uploads/2025/03/Wybrane-wyniki-BZ-KIM_dane-I-XII-2024_Gospodarstwa-domowe-w-roku-2024_publikacja-31.03.2025.pdf , dostępny 01.07.2025).
[9]. Low-code development platform. (2025-07-03). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=Low-code_development_platform&oldid=1298586561, dostępny 15.07.2025).
[10]. MIT App Inventor Community. (2025). Massachusetts Institute of Technology, (https://community.appinventor.mit.edu, dostępny 10.07.2025).
[11]. MIT App Inventor. (2025-07-07). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=MIT_App_Inventor&oldid=1299237116, dostępny 15.07.2025).
[12]. Mobile App. (2025-03-04). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=Mobile_app&oldid=1278753502, dostępny 15.07.2025).

[13]. Mobile device. (2025-07-13). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=Mobile_device&oldid=1300284096, dostępny 15.07.2025).
[14]. Mobile operating system. (2025-07-15). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=Mobile_operating_system&oldid=1300543897, dostępny 15.07.2025).
[15]. No-code development platform. (2025-07-06). W: *Wikipedia*, (https://en.wikipedia.org/w/index.php?title=No-code_development_platform&oldid=1299017369, dostępny 15.07.2025).
[16]. Reference Documentation. (2025). Massachusetts Institute of Technology, (https://ai2.appinventor.mit.edu/reference/, dostępny 10.07.2025).
[17]. Ross R., Pillitteri V. (2024). *Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations*, National Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication (SP) NIST SP 800-171r3, https://doi.org/10.6028/NIST.SP.800-171r3.
[18]. Sheldon R. (2024-02-29). *DEFINITION: mobile device*, (https://www.techtarget.com/whatis/definition/mobile-device, dostępny 01.07.2025).
[19]. Stasiewicz A. (2016). *Android Studio. Podstawy tworzenia aplikacji*, Helion, Gliwice.
[20]. StatCounter. (2025). *Mobile Operating System Market Share Worldwide*, (https://gs.statcounter.com/os-market-share/mobile/worldwide, dostępny 15.07.2025).
[21]. The MIT App Inventor Library: Documentation & Support. (2025). Massachusetts Institute of Technology, (https://appinventor.mit.edu/explore/library, dostępny 10.07.2025).
[22]. Ustawa z dnia 4 kwietnia 2019 r. o dostępności cyfrowej stron internetowych i aplikacji mobilnych podmiotów publicznych. Dz.U. 2019 poz. 848, z późn. zm., https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20190000848.