

1 Koncepcja programowania

Programowanie w systemach takich jak AI2 składa się z 2 etapów:

- tworzenie interfejsu użytkownika (elementy interfejsu aplikacji czyli komponenty, ich ułożenie i właściwości)
- obsługa zdarzeń systemowych czyli oprogramowanie tego jak wszystko ma działać.

Dokumentacja do systemu Applinventor2: [tutaj](#).

1.1 Interfejs aplikacji i hierarchia komponentów.

Podstawowym elementem składowym każdej aplikacji jest ekran (screen). Jest elementem minimalnym i obligatoryjnym. Na nim umieszcza się wszystkie inne komponenty. Niektóre z nich mogą zawierać inne komponenty w celach porządkowych lub właściwego oprogramowania. Poniżej ważniejsze komponenty i działy w jakich się znajdują.

User Interface

- Button - przycisk
- CheckBox - pole wyboru opcji
- Image - obraz
- Label - napis tekstowy tzw. etykieta
- Notifier - wyskakujące okienka dialogowe
- TextBox - pole edycyjne dla wpisywania tekstu

Layout

- HorizontalArrangement - układ innych komponentów w poziomie
- VerticalArrangement - układ innych komponentów w pionie
- TableArrangement - układ tabelaryczny MxN elementów

Media

- Camera - kamera
- Player - odtwarzacz dużych plików
- Sound - odtwarzacz dla krótkich dźwięków
- SoundRecorder - dyktafon
- TextToSpeech - syntezytor mowy

Drawing and Animation

- Canvas - obszar 2D dla animacji (tylko dla Ball i ImageSprite)
- Ball - piłeczka do animacji
- ImageSprite - poruszający się obrazek

Sensors

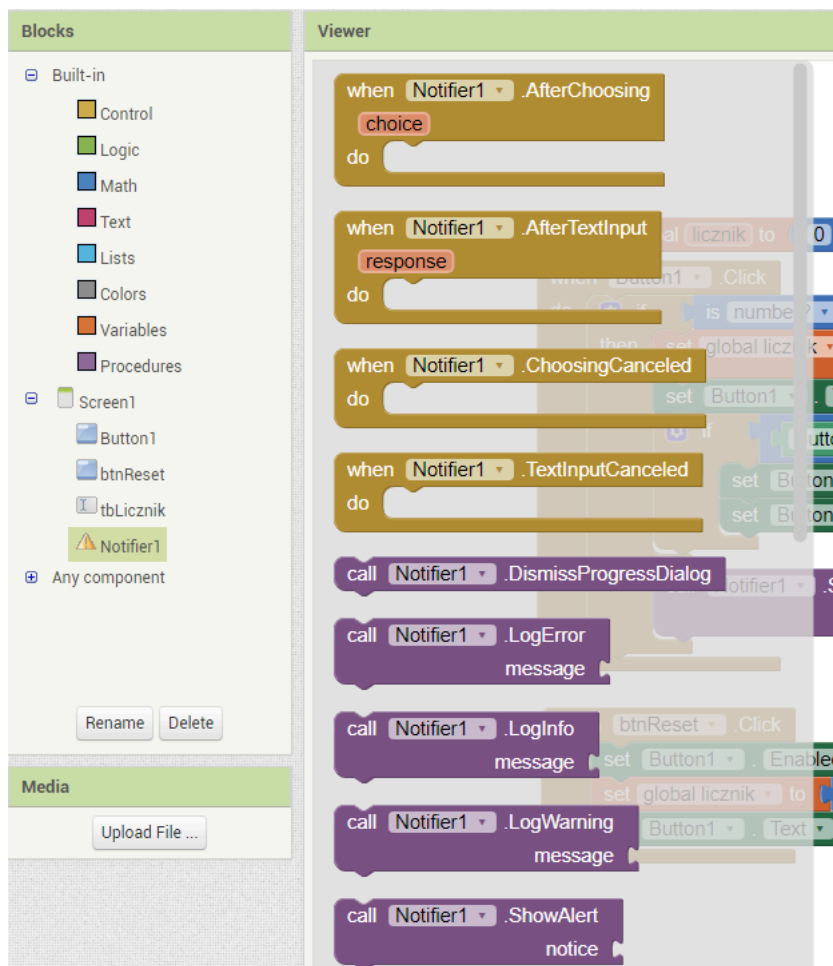
- AccelerometerSensor - akcelerometr (o ile smatrfon posiada)
- Clock - zegar

Social

- PhoneCall - uruchomienie telefonu
- Sharing - współdzielenie zasobów w mediach społecznościowych i nie tylko

I wiele innych. Każdy komponent ma swoją niepowtarzalną nazwę i różne parametry (ang. properties) zależnie od swego przeznaczenia. Komponenty mogą być widzialne (widać je wtedy na ekranie aplikacji) oraz niewidzialne (nie widać ich na tym ekranie ale działają w tle świadcząc różne usługi). Komponenty widzialne przeznaczone raczej są do bezpośredniej interakcji z użytkownikiem. Komponenty niewidzialne pełnią różne role systemowe takie jak odczyt czasu, odczyt GPS, akcelerometru i inne. Z nich raczej pozyskujemy dane do działania programu.

Komponenty oprócz posiadania właściwości reagują na zdarzenia (ang. events) przykładowo przycisk zareaguje na kliknięcie. Zdarzenia możemy oprogramować sami (np. zmiana koloru tła ekranu) lub skorzystać z gotowych metod obsługi zdarzeń. Przykład: kliknięcie przycisku obsłużymy poprzez wykorzystanie gotowej metody showAlert komponentu Notifier co spowoduje wyświetlenie okienka dialogowego. Gotowe metody obsługi zdarzeń wbudowane są w większość komponentów. Zobaczyc je możemy w oknie Blocks, po kliknięciu w dany komponent po lewej. Poniżej przykład listy zdarzeń (brązowy) i listy metod (fioletowy) związanych z komponentem Notifier.



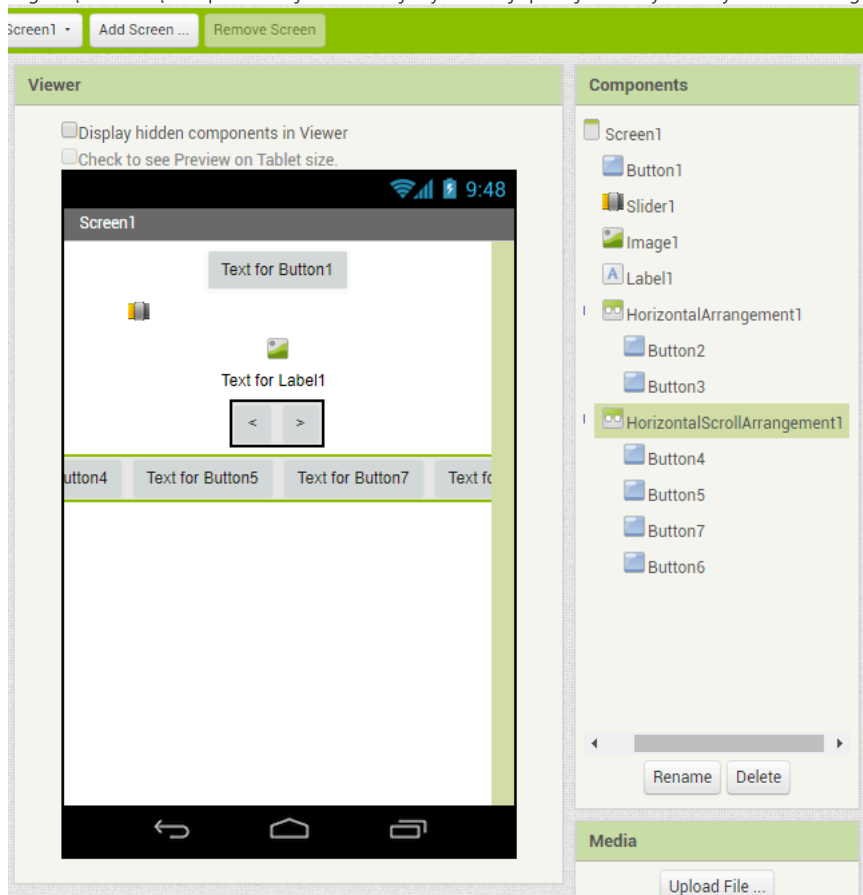
1.2 Projektowanie interfejsu aplikacji

Elementy składowe

Głównym elementem aplikacji jest ekran (screen) na którym umieszczamy wszystkie komponenty zarówno widoczne jak i niewidoczne.

Hierarchia logiczna

Logiczną hierarchię komponentów jakie zastosujemy w naszej aplikacji możemy zobaczyć w oknie Designer w sekcji Components.



Układ komponentów

Domyślny układ przyciąganych na ekran komponentów to pozycjonowanie do lewej i do góry zgodnie z kolejnością umieszczania ich na ekranie. Sam ekran ma dwie

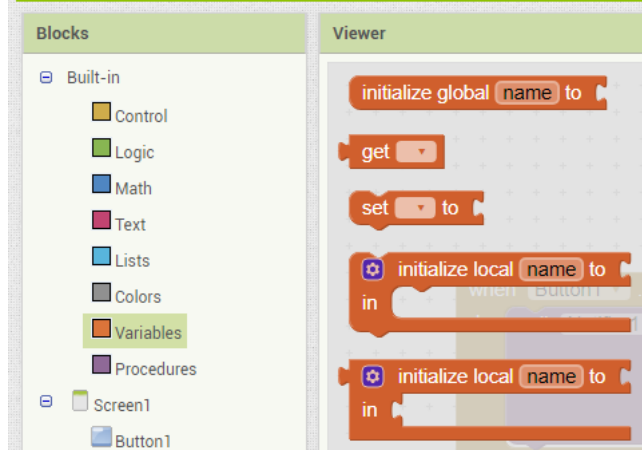
właściwości pozycjonowania wszystkich jego widocznych elementów: AlignHorizontal i AlignVertical. Ustawiać można odpowiednio do: lewej, środka, prawej oraz do góry, środka, do dołu. To pozycjonowanie dotyczy się wszystkich elementów umieszczonych bezpośrednio na ekranie. Aby pozycjonować oddzielnie od siebie poszczególne komponenty należy zastosować komponent pozycjonowania z Palette/Layout o nazwie HorizontalArrangement, VerticalArrangement, TableArrangement. Te komponenty pozwalają niejako w sobie umieszczać inne komponenty i odpowiednio je układać wobec siebie. Cały komponent pozycjonowania może także być odpowiednio ułożony na ekranie zgodnie z pozycjonowaniem ekranu omówionym nieco wyżej.

Aplikacja wieloekranowa

Aplikacja wykorzystująca wiele ekranów to w zasadzie wiele aplikacji ponieważ każdy ekran traktowany jest jak oddzielna aplikacja. Ekran można otwierać i zamykać przekazując i pobierając między nimi różne dane. Rozbudowane aplikacje posiadające wiele komponentów mogą z powodzeniem wykorzystywać kilka ekranów.
[...]

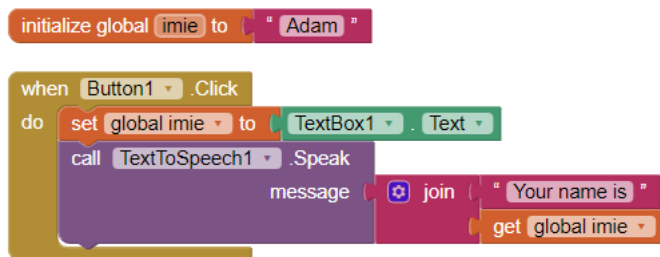
1.3 Zmienne

Zmienne w programach są pojemnikami na nasze dane. Do zmiennej można zapisać pewną wartość i podobnie ze zmiennej można taką wartość odczytać. Zmienne dzielą się na globalne dostępne w całej aplikacji oraz na zmienne lokalne istniejące tylko w danej procedurze. Zmienne dostępne są w oknie Bloków, w sekcji Blocks - Variables.



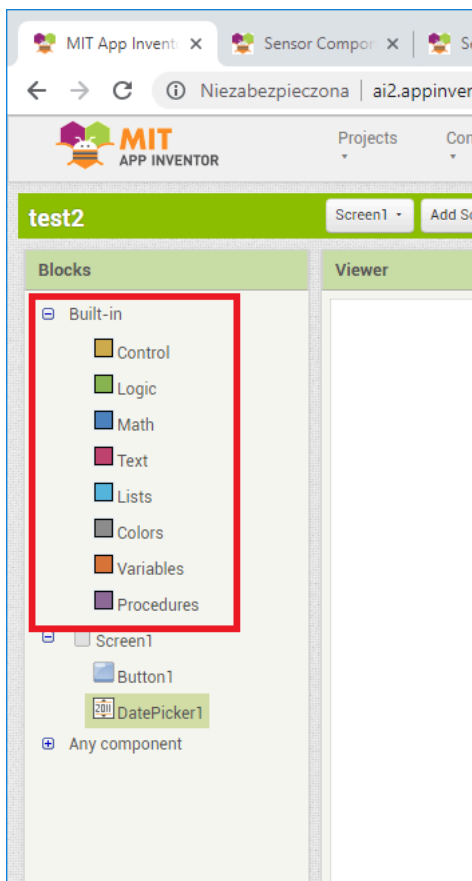
Zmienną można utworzyć (zainicjalizować), odczytać (get), ustawić (set). Jeśli piszemy program pobierający od użytkownika jego imię to zapewne będziemy go w programie używać. Warto więc powołać zmienną i jej przypisać imię użytkownika:

W powyższym przykładzie utworzono zmienną *imie* i zainicjalizowano jej wartość ciągiem znaków "Adam", *kolejno* po kliknięciu w przycisk Button1 zmiennej tej zostanie przypisana wartość z pola edycyjnego komponentu TextBox1. Kolejny krok to przekazanie połączonego tekstu "Your name is" + *imie* ze zmiennej do syntezy mowy.

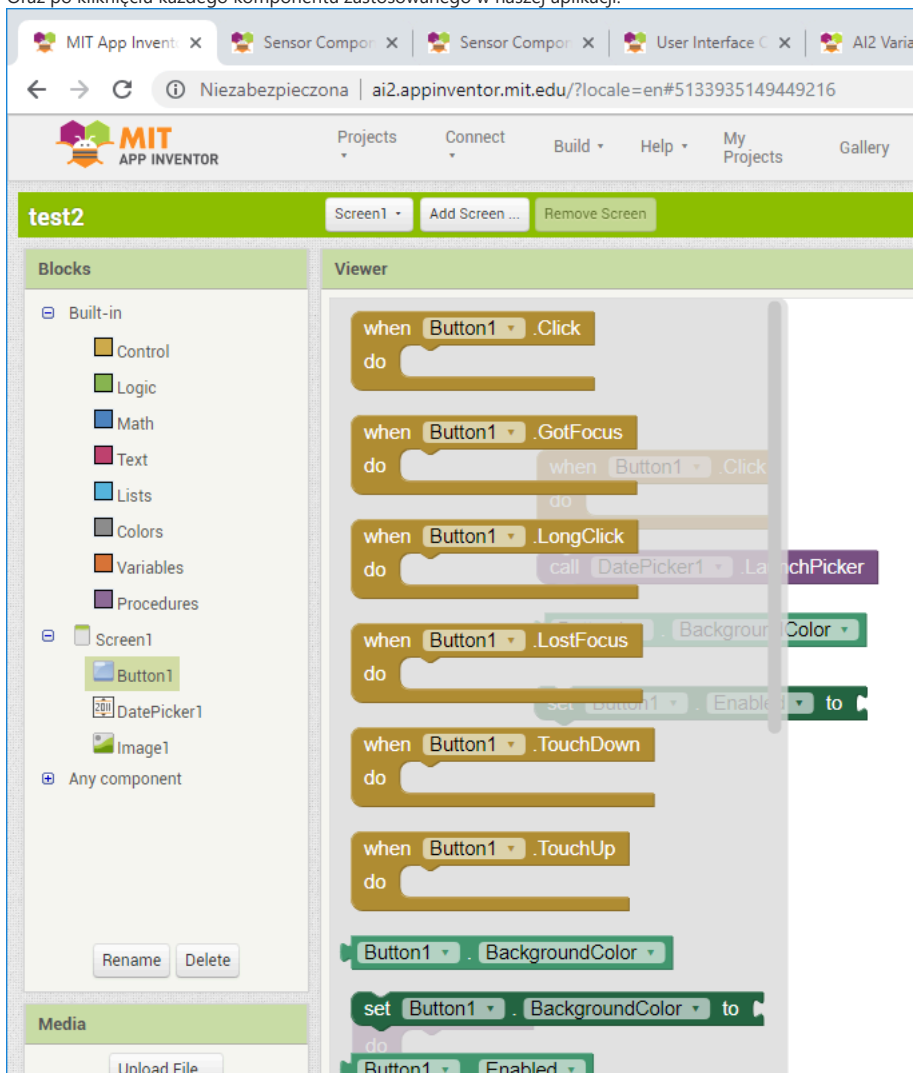


1.4 Rodzaje bloków

Rodzaje bloków znajdziemy w sekcji Bloki (ang. Blocks) okna Blocks.



Oraz po kliknięciu każdego komponentu zastosowanego w naszej aplikacji.



Rodzaje bloków:

- ogólne** (Built-in), mogące należeć do różnych komponentów
- Control - warunkowe, pętle, wywołujące pewne zdarzenia systemowe np jak zamknięcie aplikacji
- Logic - operacje logiczne and, or, porównania

Math - operacje i funkcje matematyczne

Text - obsługa tekstów

Lists - obsługa list

Colors - kolory

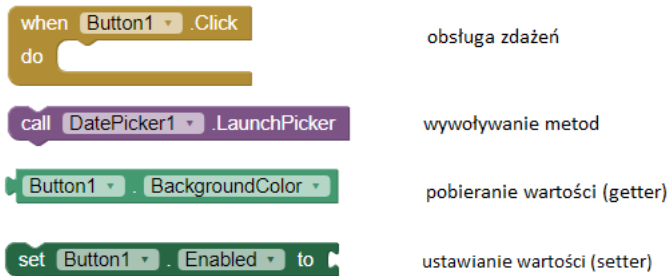
Variables - obsługa zmiennych globalnych i lokalnych

Procedures - obsługa procedur

b) związane z komponentami:

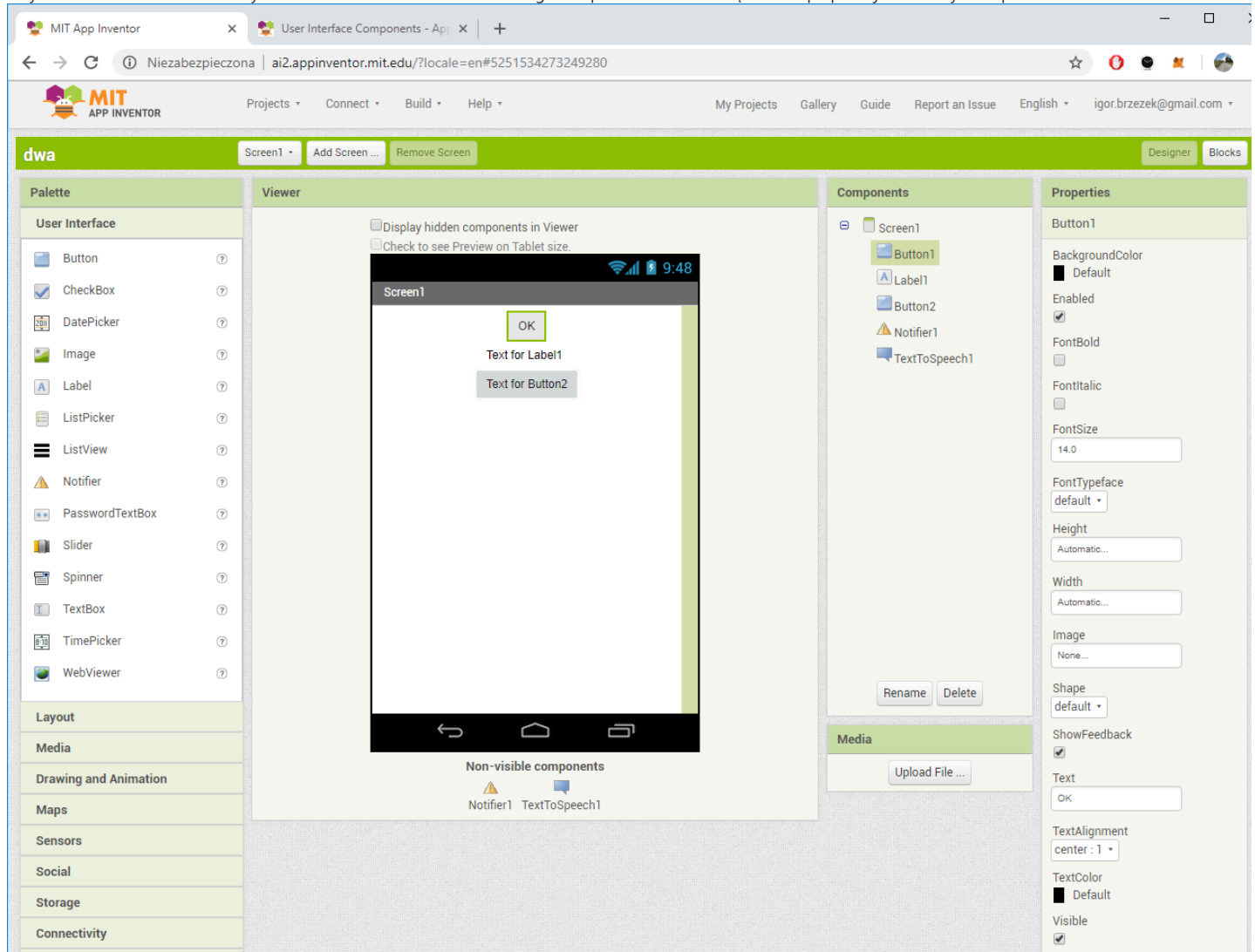
Każdy komponent ma inny zestaw bloków co jest ściśle związane z jego właściwościami, metodami jakie udostępnia inaczej mówiąc z tym do czego ten służy blok.

Przykładowo przycisk (ang. Button) będzie reagował na naciśnięcie, można zmienić jego opis, położenie itp. Komponent typu Obraz (ang Image) posiada właściwości pozwalające wczytać obraz, obrócić go itp. Każdy komponent ma swoje właściwości i metody z nim związane. Poniżej przykład dwóch komponentów: button i datepicker.



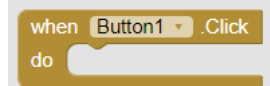
1.5 Właściwości (ang. Properties)

Właściwości to parametry komponentów takie jak napis na komponentcie, kolor, pozycja na ekranie itp. Każdy komponent ze względu na swoje przeznaczenie dysponuje innym zestawem właściwości. Wszystkie właściwości aktualnie zaznaczonego komponentu widoczne są w oknie po prawej oznaczonym Properties.



1.6 Zdarzenia (ang. Events)

Zdarzenia związane są z tym "co potrafi komponent" czyli do czego został stworzony. Blokiem obsługi zdarzenia (ang. event handler) jest poniższy blok (when ... do):



Bloki zdarzeń mają kolor brązowy i wyglądają jak wyżej. W tym wypadku jest to blok przycisku. Interpretacja jest taka: "kiedy naciśnięto przycisk Button1 wykonaj akcje ...". Akcją będzie to co uznamy za stosowne i musimy ją w jakiś sposób oprogramować. Przykładowo, kiedy naciśnięty zostanie owy przycisk zmieni się kolor tła. Komponenty mogą obsługiwać wiele różnych zdarzeń, przykładowo przycisk:

Events

Click()

User tapped and released the button.

GotFocus()

Indicates the cursor moved over the button so it is now possible to click it.

LongClick()

User held the button down.

LostFocus()

Indicates the cursor moved away from the button so it is now no longer possible to click it.

TouchDown()

Indicates that the button was pressed down.

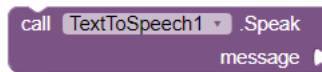
TouchUp()

Indicates that a button has been released.

Wszystkie zdarzenia komponentów opisane są w [dokumentacji](#). Zdarzenia obsługiwane są sekwencyjnie i w trakcie obsługi jednego zdarzenia nie jest obsługiwane inne. Metody mogą także wywoływać zdarzenia. Przykładowo po odczytaniu pliku (metoda ReadFrom) wywołane jest zdarzenie .GotText dzięki któremu można odczytać jego zawartość.

1.7 Metody

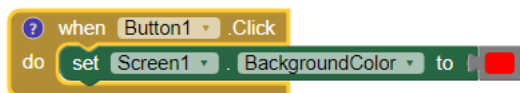
Z wcześniej wymienionymi *zdarzeniami* związane są *metody* ich obsługi. Zdarzenia można obsługiwać we własnym zakresie tworząc odpowiednią akcję i tym samym programując naszą aplikację lub korzystać z gotowych metod. Poniżej widok wbudowanej metody .Speak komponentu syntezy mowy.



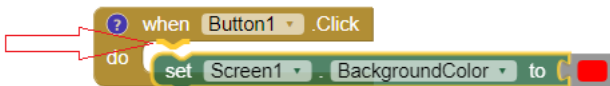
Bloki metod mają kolor fioletowy jak wyżej. Kiedy zdarzenie zostaje wywołane (np. naciśnięcie przycisku) należy je jakoś obsłużyć. Do tego stworzono metody. Inaczej mówiąc: metody obsługi zdarzeń. Różne komponenty mają różne rodzaje zdarzeń a co za tym idzie różne metody ich obsługi. Jak wynika z powyższego zdarzenia możemy obsługiwać "sami" lub korzystać z gotowych metod. Dodać należy, że zdarzenie pochodzące od jednego komponentu może wywoływać metodę należącą do innego komponentu. Przykładowo: kliknięcie przycisku może wywołać metodę wypowiedzenia tekstu przez komponent syntezy mowy.

Własna obsługa zdarzeń.

Poniżej przykład obsługi zdarzenia naciśnięcia przycisku (ang. Click). Kiedy przycisk zostanie naciśnięty zmieni się kolor tła ekranu na czerwony. Dla czego? Ponieważ tak zaplanował programista.

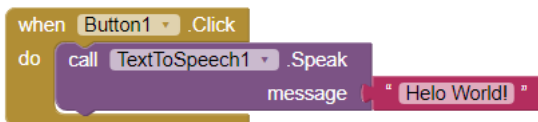


Całość zaprogramowania takiej akcji polega na przeciągnięciu odpowiednich bloków i umieszczenie "jednego w drugim". Bloki mają specjalne kształty, które pozwalają je składać tylko na jeden sposób tak więc możliwość pomyłki przy budowie aplikacji jest minimalna. Dodatkowo kiedy zbliżamy bloki do siebie pojawia się element sugerujący czy i co pasuje do siebie:



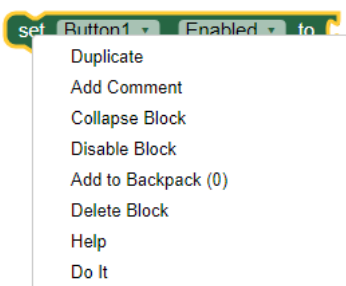
Obsługa zdarzeń na bazie wywołania gotowej metody.

Poniżej mamy przykład gdzie zdarzenie typu .Click, czyli kliknięcie w przycisk, związane z komponentem przycisk (Button) wywoła metodę .Speak związaną z innym komponentem jakim jest syntezy mowy (TextToSpeech).



1.8 Menu kontekstowe

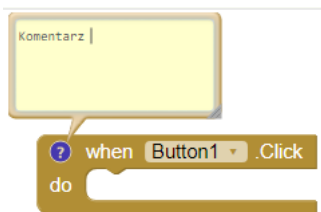
Z każdym blokiem związane jest menu kontekstowe wywoływane prawym klawiszem myszki.



Blok można:

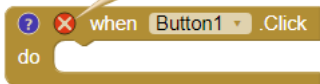
- zduplikować co przyspiesza pracę
- dodać komentarz co jest zawsze dobrym pomysłem zwłaszcza w przypadku budowy większych procedur i bardziej złożonych konstrukcji w programie
- zwinąć blok aby zwiększyć ilość miejsca na ekranie (blok działa nadal)
- zablokować blok (blok wtedy nie działa czyli tak ja by go nie było)
- usunąć blok (szybciej będzie klawiszem Del)
- wywołać stronę pomocy czyli opisu parametrów bloku
- niektóre bloki posiadają funkcję wykonania ich (Do it) co pozwoli wykonywać program krok po kroku

Jedną z bardziej przydatnych funkcji są komentarze. Po włączeniu na bloku pojawi się ikonka znaku zapytania:

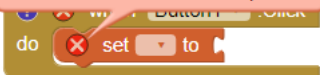


I w okienku nad blokiem można wpisywać swoje uwagi. Komentarz jest tylko dla człowieka ApplInventor pomija je. W tej samej okolicy bloku pojawić się może także symbol błędu, jeśli zrobimy coś niezgodnego z zasadami programowania.

This is a duplicate event handler for this component.



Select a valid item in the drop down.

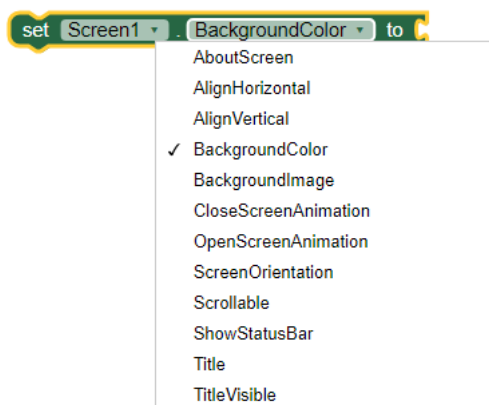


W przykładzie powyżej mamy zasygnalizowane dwa błędy:

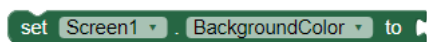
- zduplikowany blok obsługi zdarzenia
- nie wybrana zmienna do ustawienia

1.9 Pobieranie i ustawianie wartości

W ApplInventor występują elementy pozwalające pobrać wartość z komponentu zwane getter (pobieracz) (od ang. get) oraz ustawić wartość np właściwości komponentu czy wartość zmiennej zwane setter (ustawiacz) (od ang. set). W obu przypadkach z zestawu bloków wybieramy żądany komponent a następnie wybieramy potrzebną właściwość.

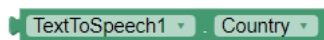


setter:



Ustaw właściwość BackgroundColor obiektu Screen1 na...

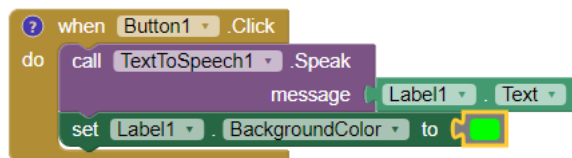
getter:



Pobierz wartość właściwości Country obiektu TextToSpeech1

Przepływ danych jak zwykle z prawej do lewej. Do elementu *setter* z prawej strony zostanie wprowadzona wartość (z innego elementu wyprowadzającego dane), natomiast *getter* z prawej do lewej przekaże wartość wybranej właściwości.

1.10 Przykład zastosowania getter, setter oraz wywołania metody obsługi zdarzenia

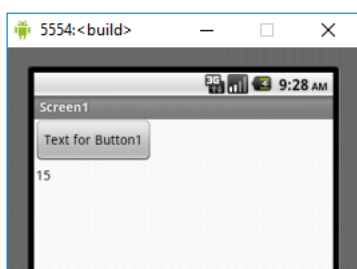


Interpretacja:

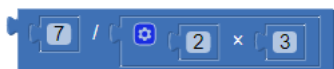
Kiedy kliknięto przycisk Button1 kolejno wywołaj metodę Speak (syntezuj tekst) syntezy mowy TextToSpeech1, metoda ta ma pobrać tekst z etykiety Label1 i wypowiedzieć go, kolejno ustaw kolor tła etykiety Label1 na zielony. Jak można zauważyć przepływ danych jest od prawej do lewej a wykonywanie operacji z góry na dół.

1.11 Obliczenia numeryczne

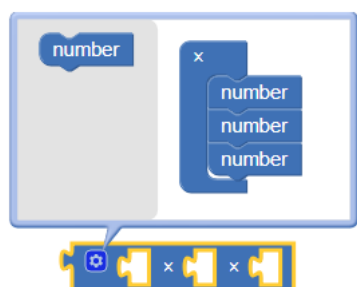
Dla wszelakich obliczeń numerycznych mamy do dyspozycji bloki w sekcji Built-in o nazwie Math. Poniżej przykład aplikacji wykonującej działanie $(2+3)*3$ i wyświetlające wynik w Label1;



Po naciśnięciu przycisku Button1 do właściwości text etykiety Label i jest wpisana (po uprzednim wyliczeniu) wartość wyrażenia $2+3*3$ z tym, że uwaga: zastosowanie podbloków działa jak nawiasy! W tym wypadku mamy: $(2+3)*3$. Jeśli chcemy zbudować wyrażenie $7/(2*3)$ to używamy dwóch bloków:



W wielu blokach znajduje się ikonka kółka zębatego. Pozwala ona na rozbudowę i modyfikację działania bloku. W przypadku bloku mnożenia:



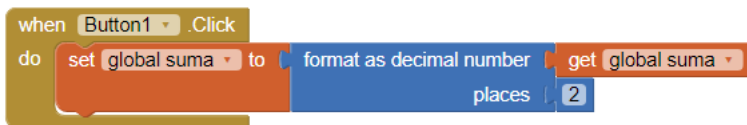
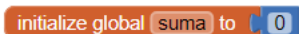
Jak widać mamy możliwość ustawiania ilości mnożonych elementów. Ciekawostka: jeśli w pustym obszarze okna programu klikniesz myszką i wpiszesz dowolną wartość np 23 a następnie naciśniesz Enter pojawi się blok z tymże numerem. Może to służyć szybkiego wstawiania elementów.

Poniżej kilka przydatnych boków z funkcjami matematycznymi:

zaokrąglanie

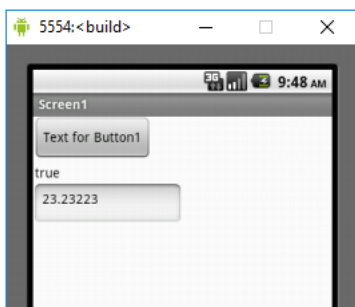
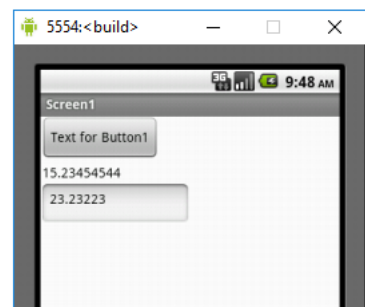
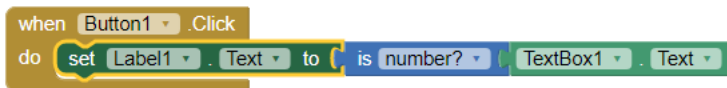


Kiedy naciśnięto przycisk Button1 tekst z Label1 został przetworzony na liczbę i o ile nią był została przeprowadzona operacja zaokrąglania do 2 miejsc po przecinku. Można oczywiście także używać zmiennych nie koniecznie elementów widocznych:



Poniżej kilka przykładów.

sprawdzanie czy wartość jest numeryczna

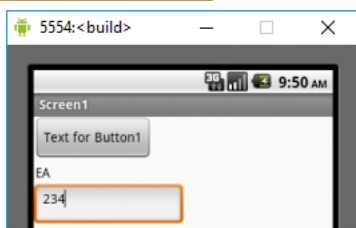


konwersja między systemami liczbowymi

```

when Button1 .Click
do set Label1 . Text to convert number base 10 to hex TextBox1 . Text

```



wartość losowa z przedziału

```

when Button1 .Click
do set Label1 . Text to random integer from 1 to 100

```

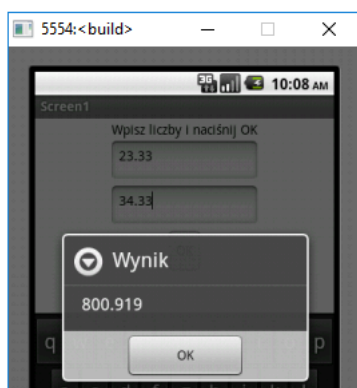
1.12 Okienka dialogowe (notifier)

Okna dialogowe to przydatne elementy pozwalające informować użytkownika o pewnych zdarzeniach w systemie, wyprowadzać wyniki obliczeń czy odpowiedzi. Poniżej przykład prostego okna dialogowego. Po wprowadzeniu dowolnych liczb do pól edycyjnych i naciśnięciu przycisku OK pojawia się okno dialogowe z wynikiem.

```

when Button1 .Click
do call Notifier1 .ShowMessageDialog
    message format as decimal number TextBox1 . Text × TextBox2 . Text
    places 3
    title "Wynik"
    buttonText "OK"

```

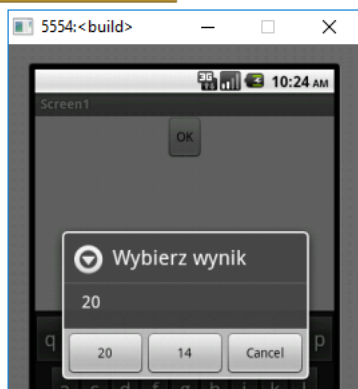


Poniżej przykład okna dialogowego z trzema przyciskami.

```

when Button1 .Click
do call Notifier1 .ShowChooseDialog
    message 4 × 3 + 2
    title "Wybierz wynik"
    button1Text "20"
    button2Text "14"
    cancelable true

```

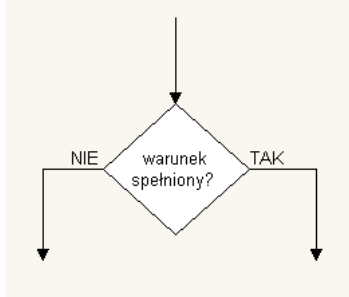


Takie okno jest w pełni interaktywne i pozwala na podejmowanie akcji w zależności od naciśniętego przycisku. Więcej na ćwiczeniach :)

1.13 Instrukcje warunkowe *if..then..else*

[Nieczo dokumentacji](#) , [Bloki decyzyjne...](#) [przykład](#)

Wybory dokonujemy wielokrotnie w trakcie każdego dnia. Obrazowo można taki wybór przedstawić tak:



Jeśli dany warunek jest spełniony zrób to a jeśli nie to zrób coś innego. Bloki warunkowe znajdują się w oknie Blocks/Blocks w sekcji Built-in w gałęzi Control:

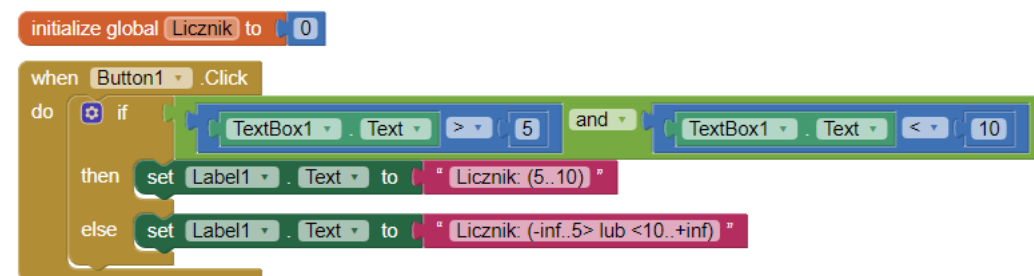


Pierwszy blok bada warunek i wykonuje akcje (można go modyfikować i rozszerzać na co wskazuje niebieska ikonka, drugi blok także przeprowadza test i zwraca wartość. Warunek *if ... then ...*

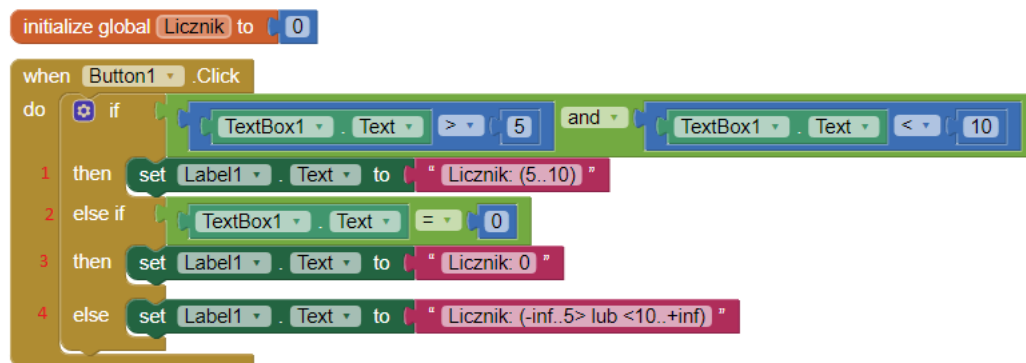
Jeśli warunek zostanie spełniony wtedy zostaną wykonane instrukcje wewnątrz warunku. Jeśli warunek nie jest spełniony instrukcje wewnątrz pętli nie zostaną wykonane a aplikacja przejdzie do wykonania dalszej części bloków.

Warunek *if ... then ... else ...*

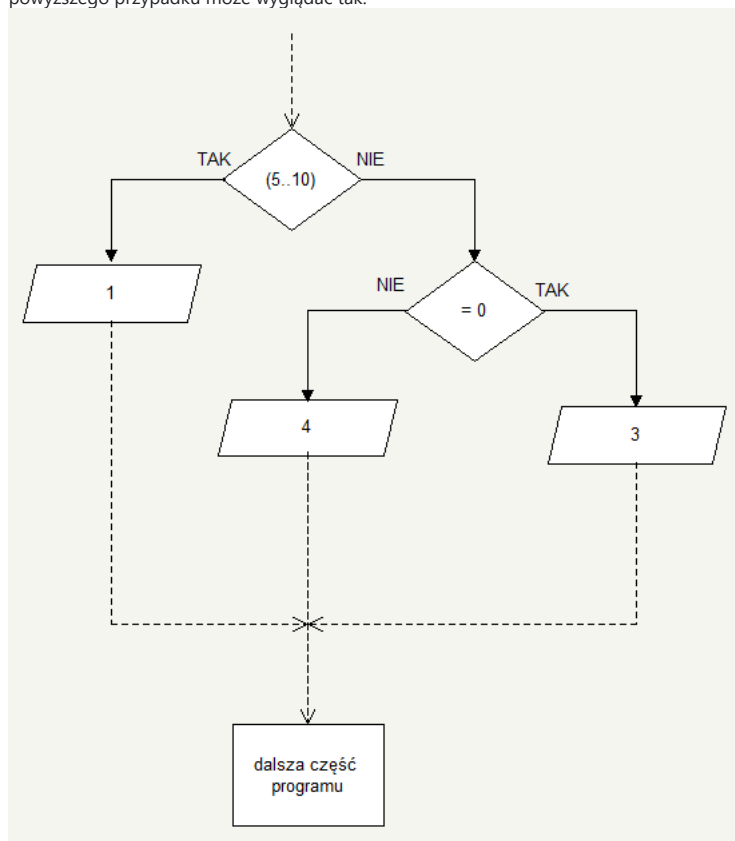
Warunek *if* można rozbudować i to znacznie pozwalając utworzyć zagnieżdżone struktury testowania warunków. Poniższy przykład ilustruje dość prostą konstrukcję: jeśli wartość z pola tekstowego jest > 5 i jednocześnie jest < 10 (czyli mieści się w przedziale otwartym $(5..10)$) zostanie wyświetlona informacja z sekcji *then* jeśli jednak jest inaczej (jakakolwiek inna wartość) zostanie wyświetlona informacja z sekcji *else*. W tym wypadku zawsze coś zostanie wykonane (o ile w polu tekstowym jest wartość numerycznie poprawna!).



Poniżej nieco bardziej rozbudowana konstrukcja.



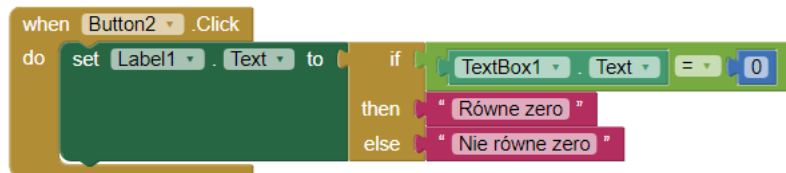
Najpierw jest sprawdzany Licznik. Jeśli jest on > 5 i jednocześnie < 10 zostanie wykonana sekcja 1, jeśli jednak tak nie będzie zostanie wykonana sekcja 2 czyli kolejny test tym razem "czy wartość z pola tekstowego = 0?". Jeśli tak to wykonana będzie sekcja 3 a jeśli nie (czyli nie jest zerem) wykonana będzie sekcja 4. Schemat blokowy do powyższego przypadku może wyglądać tak:



AI2 udostępnia jeszcze jeden ciekawy blok warunkowy.

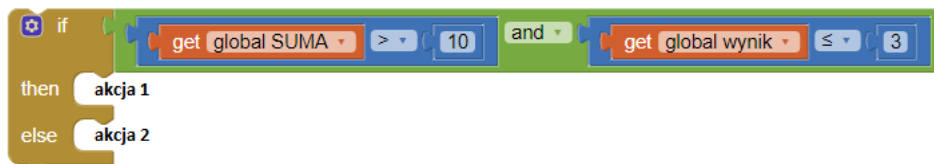


Blok ten służy do testowania warunku i w zależności od wyniku testu wyprowadza wartość zmiennej, pola czy inną do bloku nadrzędnego. Poniższy przykład: jeśli wartość z pola tekstowego równa się zero do

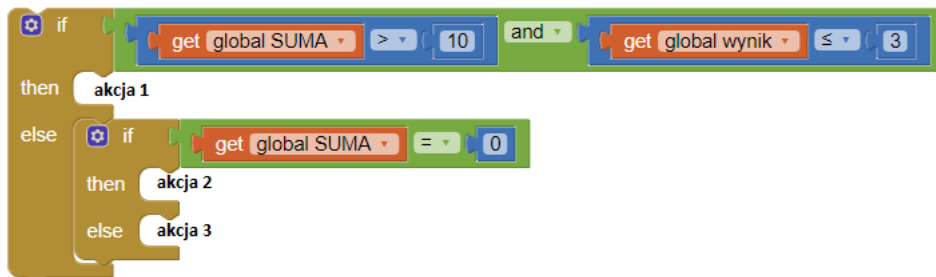


Bloki warunkowe stosuje się zazwyczaj z operacjami logicznymi. Przykład:

jeśli zmienna *SUMA* jest większa od 10 **ORAZ** (and) zmienna *wynik* jest mniejsza lub równa 3
wtedy wykonaj akcja 1
w przeciwnym razie wykonaj akcja 2.



Instrukcje warunkowe można zagnieżdżać nie należy z tym jednak przesadzać ponieważ może stać się to w pewnym momencie nieczytelne.



jeśli $SUMA > 10$ ORAZ $wynik \leq 3$

wtedy wykonaj akcja 1

w przeciwnym razie

jeśli $SUMA = 0$

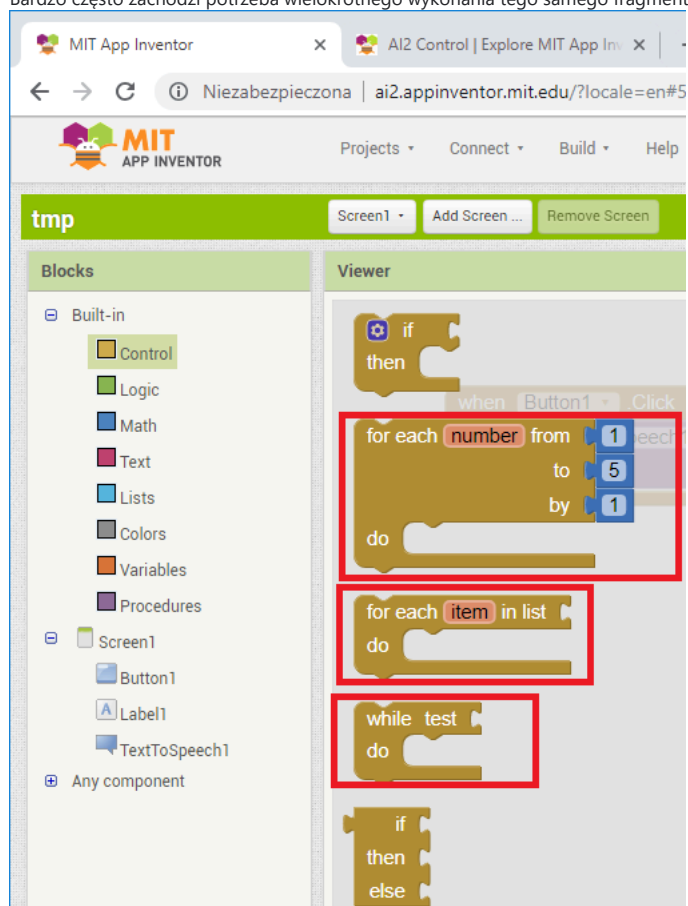
wtedy wykonaj akcja 2

w przeciwnym razie wykonaj akcja 3

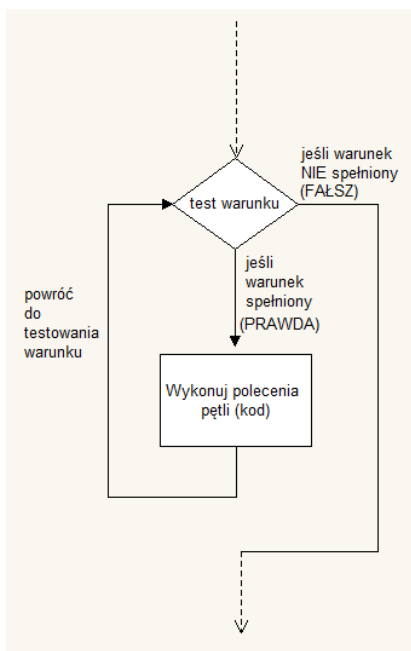
1.14 Pętle czyli powtarzanie fragmentów kodu

Bloki pętli

Bardzo często zachodzi potrzeba wielokrotnego wykonania tego samego fragmentu kodu. Służą do tego pętle, w AI2 znajdujące się w tej samej sekcji co warunki.



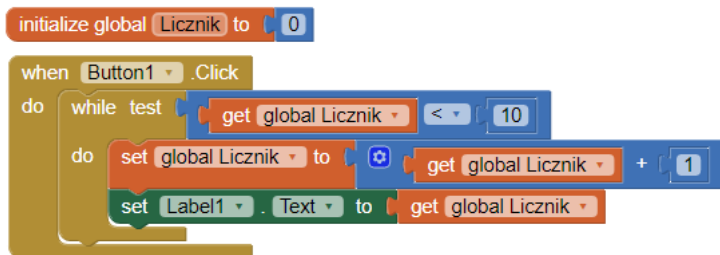
Pętla testuje warunek i jeśli jest prawdą wykonuje zawartość pętli a jeśli nie jest prawdą (lub przestał nią być) kończy działanie i przechodzi do następnych bloków o ile zostały zdefiniowane. Schemat działania pętli: (przerywana linia oznacza poprzednie polecenie programu i następne)



W AI2 jak i w innych językach programowania mamy kilka rodzajów pętli.

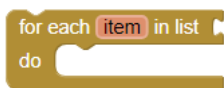
Blok: *while test ... do*

Blok ten jest typową pętlą jak na powyższym schemacie blokowym. Pętla najpierw sprawdza warunek, jeśli jest spełniony wykonuje polecenia (bloki) zawarte w pętli. Jeśli warunek nie jest spełniony bloków zawartych w pętli nie wykona ale od razu przejdzie do wykonania następnych poleceń jeśli takowe istnieją. W poniższym przykładzie mamy pętlę *while test ... do*, pętla sprawdza czy licznik jest < 10 (a tutaj jest bo ustawiona była zmienna globalna inicjalizowana na początku programu) i w tym wypadku przechodzi do wykonania poleceń (bloków) zamieszczonych w pętli czyli za każdym przebiegiem pętli (iteracją) zwiększa licznik o 1 i wyświetla tą wartość w etykiecie na ekranie. Kiedy licznik osiągnie wartość 10 warunek nie jest spełniony i pętla kończy pracę.



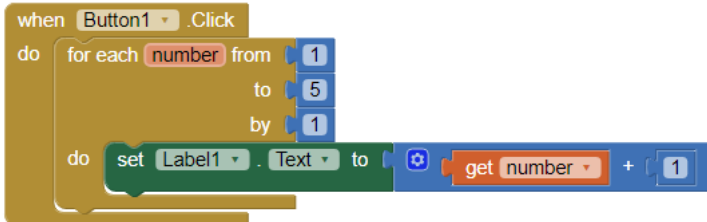
Blok *for each item in list ... do ...*

Blok ten jest specyficzny dla AI2 i jest elementem obsługującym listy a konkretnie wykonuje polecenia zawarte w pętli dla każdego elementu listy. Inaczej mówiąc pobiera kolejne elementy listy jako argumenty i przekazuje do bloków we wnętrzu pętli. Brak tu wprowadzić warunku ale jest to pętla której ilość iteracji uzależnia ilość elementów listy.



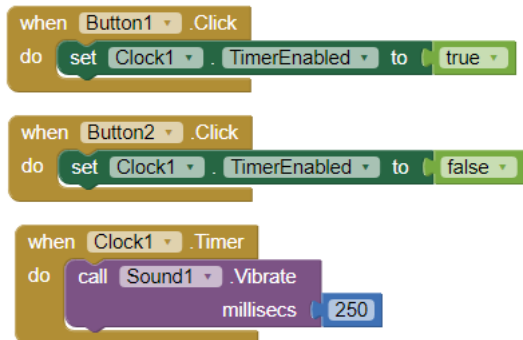
Blok *for each number from .. to .. by .. do ...*

Pętla wykona się konkretna ilość razy. W pętli wbudowany jest licznik, zmienna o nazwie *number*. Pętla startuje od wartości licznika z pola *from* liczy do wartości z pola *to* a za każdym przebiegiem pętli (iteracją) licznik ten jest zwiększany o wartość z pola *by*. Jak widać licznik można zwiększać o dowolną wartość np o 15 i podobnie start i koniec także może być dowolną wartością. W poniższym przykładzie pętla wykona się dokładnie pięć razy a licznik będzie uzyskiwał kolejne wartości: 1,2,3,4,5.



1.15 Zegar (Clock)

Zegar (sekcja Sensors) jest komponentem niewidocznym i służy do odmierzania określonych interwałów czasu. Nie jest to zegar czasu rzeczywistego podający aktualną godzinę choć może do tego służyć. W zależności od ustawienia jego właściwości *TimeInterval* (w milisekundach) wyzwalane jest co określony czas zdarzenie *Clock.Timer*. Dzięki temu można co interwał czasu określony właśnie w *TimeInterval* uruchamiać procedury, metody wbudowane lub inne bloki. Przykładowo można co 5 sekund odgrywać krótki dźwięk, wibrować telefonem itp. Sterowanie tym czy zegar działa czy nie (czyli czy co określony czas wyzwala zdarzenie *.Timer* odbywa się za pomocą parametru *TimerEnabled*. Zegar jest szczególnie użyteczny w grach i aplikacjach multimedialnych gdzie zachodzi potrzeba wywołania pewnej metody lub wykonania pewnych bloków co ściśle określony czas. W poniższym przykładzie po kliknięciu w pierwszy przycisk działania (odliczanie) zegara jest włączane i co określony w jego właściwościach interwał czasu wyzwalane jest zdarzenie *.Timer*, które w tym wypadku jest obsługiwane za pomocą metody wibrowania komponentu *Sound*.

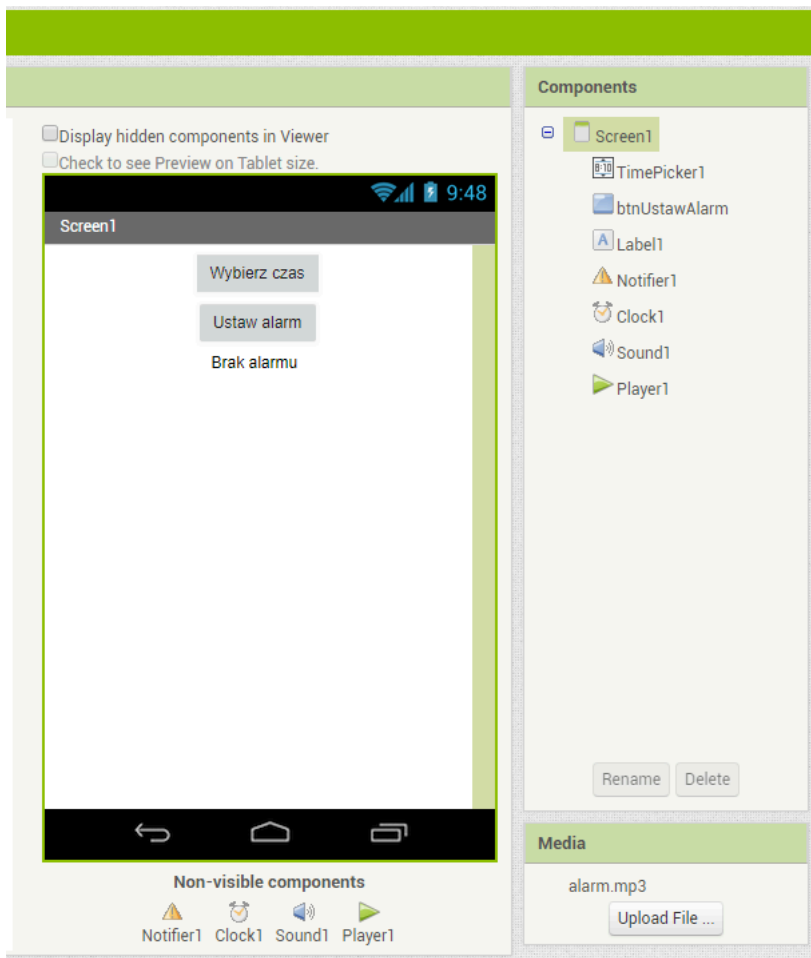


1.16 Multimedia

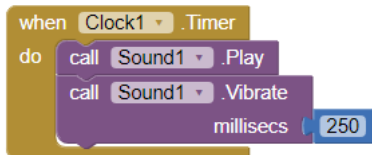
AI2 i jego aplikacje obsługują multimedia za pomocą wbudowanych komponentów Sound i Player. Należy wziąć pod uwagę, że aplikacja nie może mieć więcej niż 10MB z tego powodu większe pliki muszą być umieszczane "na zewnątrz", może to być karta SD smartfonu lub plik z adresu URL.

Komponent sound nadaje się do odtwarzania małych i krótkich plików, dla plików dużych stosować należy komponent Player. Więcej na temat mediów [tutaj](#).

Poniżej przykładowa aplikacja wykorzystująca komponenty Clock i Sound.

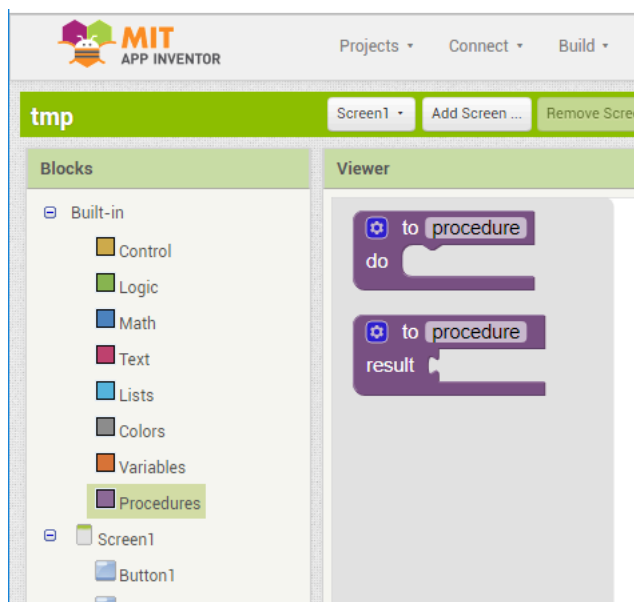


Po wyzwoleniu zdarzenia zegara `.Timer` czyli co zdefiniowany czas wywoływana jest metoda `.Play` komponentu sound odtwarzająca zdefiniowany plik audio oraz druga metoda `.Vibrate` uruchamiająca wibrator telefonu na określonej ilości ms.

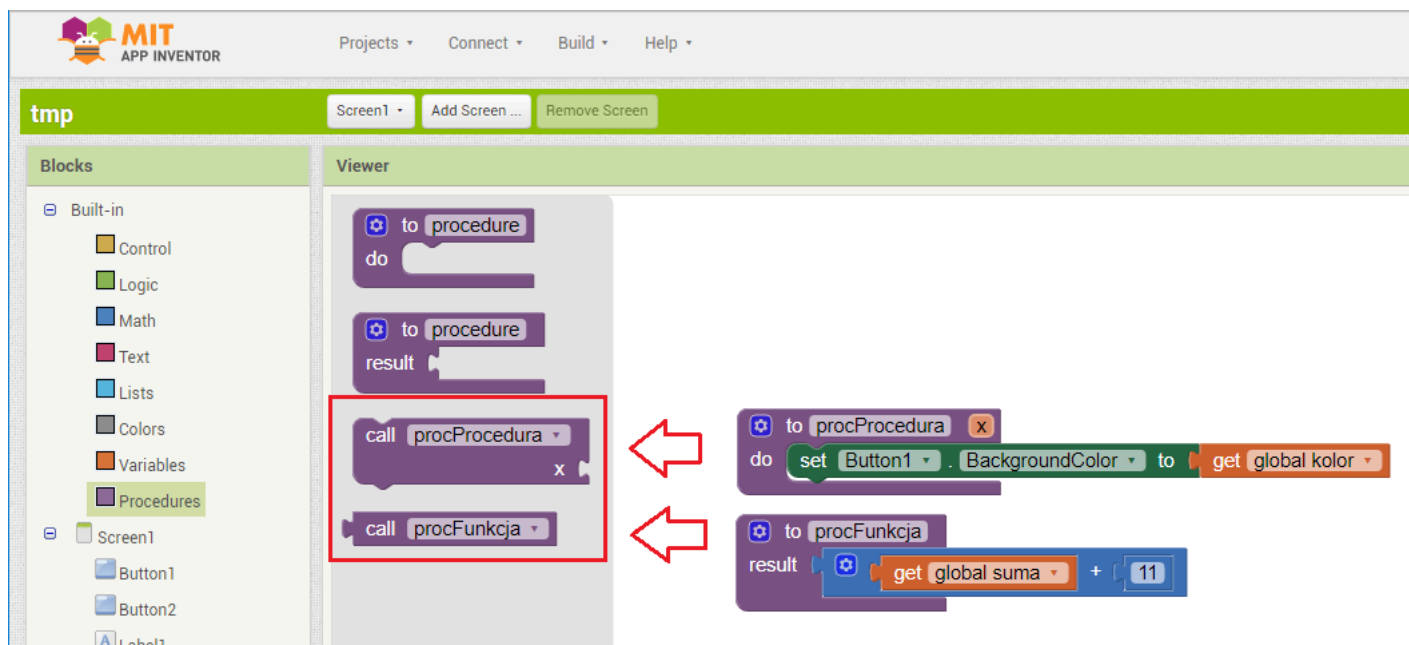


1.17 Procedury

Jak zostało to zauważone w poprzednich sekcjach procedura to zestaw logicznie powiązanych ze sobą operacji. Procedure tworzymy w programie po to aby ją wielokrotnie wykorzystywać. Raz napisany kod (w AI2 zestaw bloków) będzie mógł być wielokrotnie używany w wielu miejscach programu za pomocą operacji *wywołania* procedury. W niektórych językach programowania istnieją dwa podobne pojęcia procedura i funkcja. Procedura może coś wykonać ale nie zwraca żadnej wartości, funkcja może wartość zwracać. To podobnie jak znane ze szkoły funkcje matematyczne np $\sin(x)$ (sinus od x). Do funkcji wprowadzamy pewną wartość (tu oznaczona ogólnie jako parametr czy argument o nazwie x) a funkcja wewnątrz "coś" wykona i zwróci wartość. W AI2 nie ma rozgraniczenia na procedury i funkcje są po prostu procedury z tym, że dwa rodzaje: procedura zwykła i procedura zwracająca wartość (ang. result). Procedury dostępne są w oknie Blocks, sekcja Blocks (po lewej) w gałęzi Procedures.

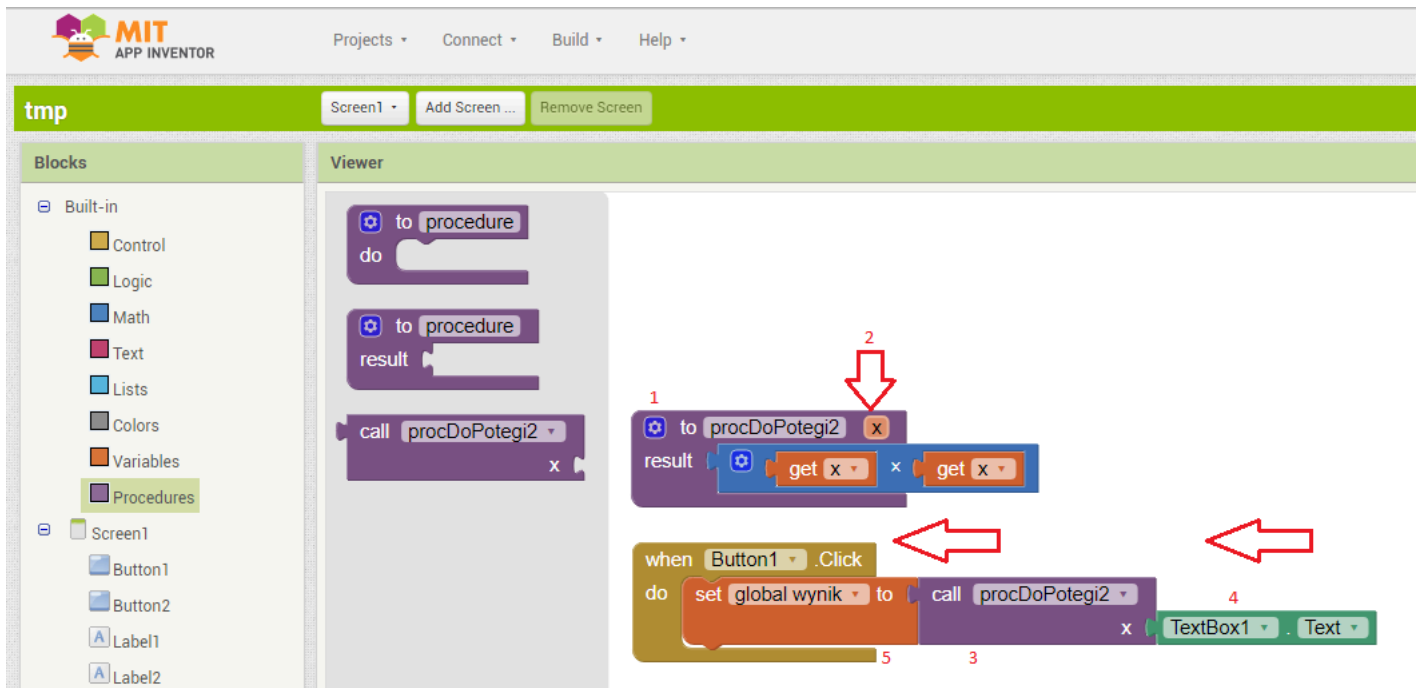


Górny blok to procedura, która tylko coś wykona, dolny blok to procedura która dodatkowo może zwracać wartość czyli coś jak funkcja. W momencie kiedy w aplikacji zastosujemy procedury w powyższej sekcji pojawiają się nowe bloki wywołania (ang. call) procedury:



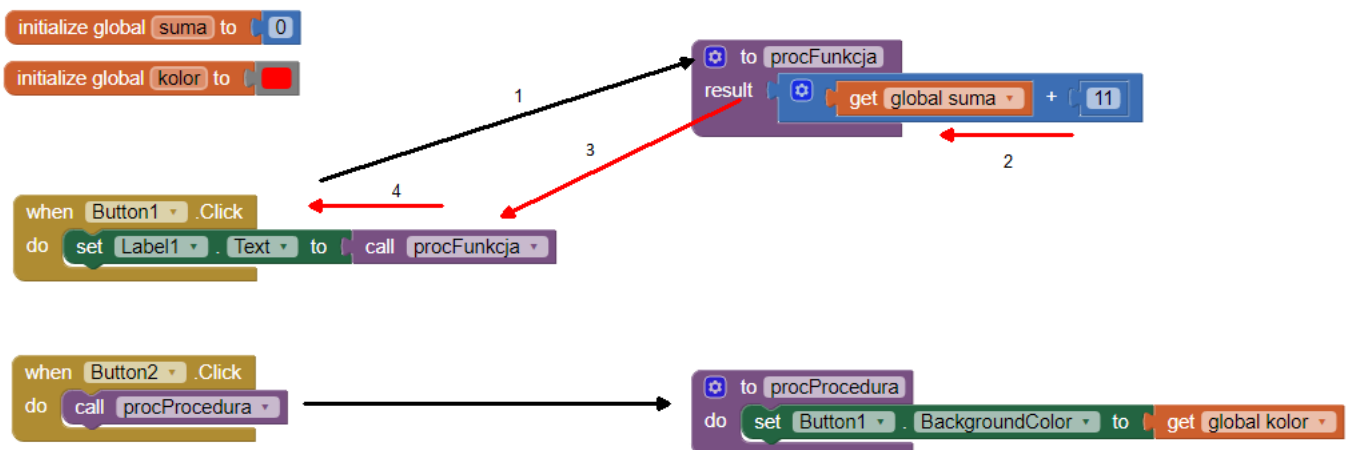
Pierwszy blok związany jest z procedurą nie zwracającą wartości natomiast drugi z procedurą mogącą zwracać wartości. Blok ten jest podobny do bloków "getter" czyli pobieraczy danych. Taka procedura zatem coś zrobi np. coś policzy i wynik zwróci do bloku który ją wywołał.

Dodatkowo do procedur obu rodzajów można przekazywać parametry inaczej mówiąc argumenty. W powyższym przykładzie do procedury o nazwie procProcedura jest przekazywany argument o nazwie x choć w tym wypadku nigdzie nie jest wykorzystywany. Poniżej nieco bardziej złożony przykład.



W powyższym przykładzie zdefiniowano procedurę o nazwie "ProcDoPotegi2" (1) zwracającą rezultat obliczeń wykonywanych wewnątrz niej. Procedura ta przyjmuje także jeden argument (parametr) zwany tutaj zmienną lokalną "x" (2). Zmienna ta jest do wykorzystania tylko wewnątrz procedury i dla tego jej typ to lokalna. Procedura przyjmuje argument, wykonuje na nim obliczenia (w tym wypadku mnoży przez siebie) i tak uzyskany wynik zwraca. Procedurę wywołuje blok "call procDoPotegi2" (3). Blok ten wywołując ją przekazuje do niej parametr w tym wypadku zawartość pola tekstowego komponentu TextBox1 (4). Procedura zostaje wywołana, uruchamia się i zwraca wynik (rezultat) do zmiennej "wynik" (5). Przepływ danych zaznaczono strzałkami poziomymi: do uruchamianej (wywoływanej) procedury przekazany jest argument (zawartość pola tekstowego) a kolejno procedura zwraca wynik obliczeń do zmiennej "wynik"

Sposoby wywołania procedur i przepływ informacji.



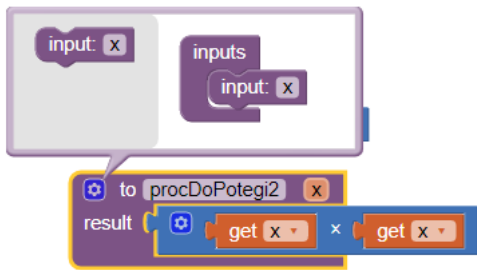
Procedura ze zwracaniem rezultatu

Kliknięcie przycisku "Button1" powoduje, za pomocą bloku *Call procFunkcja*, wywołanie (uruchomienie) zdefiniowanej wcześniej procedury o nazwie "procFunkcja" (1). Ta procedura (tutaj bez parametrów) wykonuje obliczenia (2) i tak uzyskany wynik zwraca (3) do bloku "ustawiacza", w tym wypadku opisu etykiety tekstowej o nazwie "Label1" (4).

Procedura bez zwracania rezultatu

Kliknięcie przycisku "Button2" powoduje wywołanie procedury bez parametru o nazwie "procProcedura". Procedura wykonuje się i tyle. Dokładnie wygląda to tak: procedura jest wywołana. Odbywa się skok do miejsca w pamięci gdzie znajduje się kod procedury. Kod ten jest wykonany a po zakończeniu działania procedury sterowanie wraca do miejsca jej wywołania celem kontynuacji działania programu.

Jak dodać parametr do procedury? Należy kliknąć ikonkę kółka zębatego w lewym górnym rogu bloku procedury i wstawić pożądaną ilość parametrów wejściowych oznaczonych jako *input*.



W większych aplikacjach procedury mają ogromne znaczenie, upraszczając tworzenie programu poprzez dzielenie go na funkcjonalne bloki. Można stworzyć procedurę często wykonywanych obliczeń matematycznych, logicznych lub innych czynności np resetowanie zmiennych, kolorów przycisków itp.