Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa magisterska

PROGRAMOWANIE WIZUALNE URZĄDZEŃ MOBILNYCH

Jakub Bręk

Promotor
Rafał Różycki, Dr. Hab.

Poznań, 2014 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Programowanie

Definicja programowania podawana przez literaturę przedmiotu zawiera w sobie między innymi proces tworzenia aplikacji . W takim rozumieniu programowanie jest powszechnie kojarzone przez nieprofesjonalnego odbiorcę z wielkimi ilościami kodu napisanego przez programistów. W większości przypadków tak faktycznie się dzieje. Jednak sposób pisania kodu może się różnić. Można bowiem wydzielić poziomy programowania od bardzo niskiego, wykorzystującego stosunkowo nieskomplikowane języki, do wysokiego, wymagającego wysoce specjalistycznej wiedzy i umiejętności. Przy językach niskiego poziomu, do których zalicza się np. Assembler, operowanie odbywa się na rejestrach procesora. Operacje charakteryzują się dużą szybkością, jednak napisanie bardziej skomplikowanego zadania zajęłoby ogromną ilość linii kodu oraz wymagałoby nieadekwatnego nakładu pracy. Natomiast języki wyższego poziomu wykorzystują składnię ułatwiającą zrozumienie kodu programu przez osoby, które mają z tym kodem styczność.

W celu jeszcze lepszego zrozumienia pisanego kodu powstała dodatkowa warstwa abstrakcji, gdzie zasadniczo nie jest wymagana od programistów ani jedna linijka kodu. Jest to programowania wizualne. Główne źródło tworzonego oprogramowania stanowią tutaj bloki graficzne i połączenia między nimi.

1.1.1 Programowanie wizualne

Programowanie wizualne jest to programowanie, które pozwala użytkownikowi tworzyć programy poprzez manipulację elementów graficznie, zatem inaczej niż w większości przypadków, gdy wykorzystywane są edytory tekstowe. Prawie wszystkie możliwe do osiągnięcia akcje mogą zostać zrealizowane tylko za pomocą myszki.

Jednym z narzędzi, które pozwala tworzyć aplikacje wizualnie jest App Inventor. Za pomocą powyższego programu istnieje możliwość tworzenia aplikacji na system operacyjny android. Są to głównie telefony i tablety. App Inventor jest aplikacją internetową, dostępną z poziomu przeglądarki. Nie potrzebujemy dodatkowego środowiska do tworzenia programów. App Inventor jest aplikacją stworzoną przez Google, a aktualnie utrzymywaną przez uniwersytet Massachusetts Institute of Technology (MIT). Wszystkie nowe osoby, które chciałyby zacząć programować i tworzyć oprogramowanie na system operacyjny Android mogą zacząć od App Inventora. Tworzenie aplikacji jest intuicyjne dzięki graficznemu interfejsowi, który umożliwia użytkownikowi akcje typu *przeciągnij i upuść* na interesujących go obiektach.[1] Są to proste czynności nie wymagające od użytkownika specjalistycznej wiedzy informatycznej. Nawet osoby, które nigdy nie miały do czynienia z programowaniem, nie powinny mieć większych problemów z napisaniem aplikacji.

1.1.2 Programowanie natywne

Programowanie natywne jest programowaniem na daną platformę, a więc napisane programowanie będzie na niej działać bez konieczności zainstalowania (wykorzystania, pracy) dodatkowych programów. W przypadku systemu Android jest to język Java, czyli język obiektowy wysokiego poziomu. Jest to język obiektowy wysokiego poziomu. Po napisanu programu, kod zostaje kompilowany do kodu bajtowego, którym zajmuje się maszyna wirtualna Javy (JVM). Ładuje pliki do pamięci, a następnie uruchamia zawarty w nich kod. Jednak Android nie posiada JVM. Zamiast JVM, Google wyposażył Android w maszynę Dalvik'a. Dalvik jest to maszyna wirtualna,

przystosowana specjalnie do urządzeń mobilnych, gdzie szczególną uwagę należy zwrócić na małe zasoby pamięci, energii i niewielką prędkość procesorów. Kod bajtowy stworzony przez kompilator nie jest w 100% kompatybilny z kodem bajtowym Javy. Nie można tutaj korzystać z bardziej zaawansowanych cech jakimi są Class Loadery czy Java Reflection API. [2]

1.2 Cel i zakres pracy magisterskiej

Celem pracy magisterskiej jest porównanie tworzenia aplikacji na platformę android przy pisaniu aplikacji w języku Java oraz przy wykorzystaniu narzędzia oferowanego online - App Inventor. Praca zawiera porównanie tworzenia oprogramowania z różnych perespektyw, między innymi takich jak:

- Czas potrzebny na stworzenie aplikacji.
- Możliwości jakie daje nam App Inventor, jakich rzeczy tam brakuje, a co można użyć.
- Łatwość stworzenia aplikacji.
- Porównanie takich samych aplikacji pod względem zużycia procesora oraz pamięci.
- Porównanie wydajności tych samych algorytmów pod względem czasu.
- Możliwość stworzenia bardziej zaawansowanej aplikacji korzystającej z wielu funkcji telefonu
- Ewentualna konieczność dodatkowych narzędzi potrzebnych do tworzenia aplikacji

Dzięki takiemu porównaniu powinien wyłonić się bardziej wyrazisty obraz narzędzia, jakim jest App Inventor. Osobom wstępnie zainteresowanym programowaniem, np. gimnazjalistom i licealistom ułatwi decyzję odnośnie wyboru ścieżki nauki: czy warto poznać język ? App Inventor, czy też od razu (osobno!) uczyć się języka? Java i zyskać dostęp do wszystkich funkcji Androida. Nauczanie podstaw programowania poprzez tworzenie aplikacji na system Android mogą także rozważyć nauczyciele informatyki.

W pracy zostały również przedstawione wady oraz zalety pisania oprogramowania przy wykorzystaniu App Inventora. Programowanie wizualne, mimo że wydaje się łatwiejsze, niesie ze sobą pewne niedogodności. Pewnych rzeczy prawdopodobnie nie da się zrealizować, a pewne są możliwe do zrealizowania w sposób o wiele prostszy.

1.2.1 Struktura pracy magisterskiej

W rozdziale 2 przedstawiono podstawowe pojęcia zastosowane w redagowaniu pracy magisterskiej. Terminy te zostały wyjaśnione, aby bez problemu zrozumieć bardziej skomplikowane zagadnienia. Opisane są tutaj główne komponenty wchodzące wchodzące w skład tego rozdziału, a także inne narzędzia, o których jest mowa w późniejszych rozdziałach.

W rozdziale 3 zawarto teorię dotyczącą App Inventora. Opisana jest tutaj między innymi architektura aplikacji.

W rozdziale 4 pokazano zastosowane podejście do rozwiązania problemu. Opisano tutaj w jaki sposób stworzone aplikacje były testowane oraz w jaki sposób wykorzystane zostało narzędzie Dalvik Debug Monitor.

W rozdziale 5 przedstawiono wyniki uzyskane podczas pisania pracy magisterskiej. Są to stworzone aplikacje, które testują narzędzie App Inventor pod różnym kątem. W zakończeniu rozdziału zestawiono zalety oraz wady, zarejestrowane na podstawie pisania powyższych aplikacji.

W rozdziale 6 przedstawiono wnioski uzyskane po napisaniu pracy magisterskiej. Można znaleźć tutaj informacje dla kogo skierowany jest App Inventor oraz czego można od niego oczekiwać.

Rozdział 2

Podstawowe pojęcia

W niniejszym rozdziale zostaną zawarte podstawowe pojęcia i mechanizmy używane przez aplikację App Inventor. Ideą jest tutaj przybliżenie ważnych terminów informatycznych.

2.1 App Inventor

W grudniu 2013 została wydana druga wersja systemu App Inventor. Starszą wersję określono jako Classic. Oba narzędzia są do siebie bardzo podobne, jednak projektów stworzonych w pierwszej wersji systemu nie można zaimportować do wersji nowszej. W niniejszej pracy magisterskiej skupiono się na nowszej wersji App Inventora.

App Inventor jest systemem pozwalającym na tworzenie aplikacji poprzez używanie jedynie przeglądarki internetowej. Jest to zatem aplikacja internetowa umożliwiająca zredagowanie programu informatycznego nawet przez użytkowników dysponujących niewielkim zasobem profesjonalnej wiedzy i umiejętności z zakresu programowania.

Potencjał aplikacji jest bardzo duży. Można to stwierdzić obserwując ilość aktywnych użytkowników. W maju 2014 ich liczba wynosiła 87tys. tygodniowo. Liczba zarejestrowanych użytkowników to 1,9mln w 195 krajach świata. Stworzyli oni łącznie 4,7mln projektów.[And]

2.2 Główne komponenty

App Inventor celowo ułatwia programowanie poprzez wizualizację tworzonych komponentów i intuicyjny interfejs. App Inventor składa się z 3 głównych komponentów, jakimi są:

- App Inventor Designer
- App Inventor Blocks Editor
- Android Device Emulator

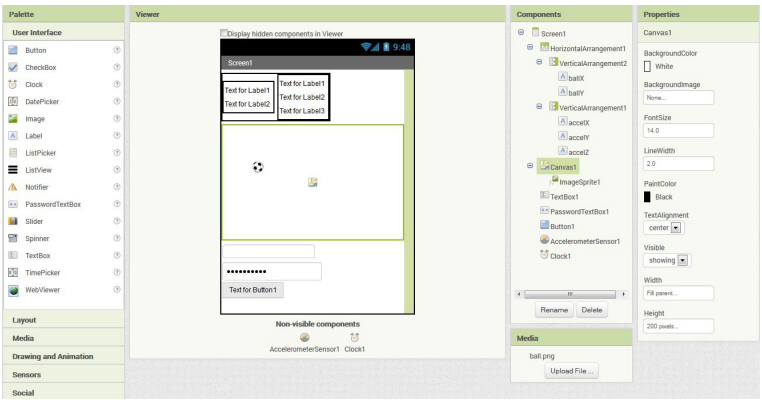
2.2.1 App Inventor Designer

Jednym z głównych widoków których można używać jest widok Designera. Projektowanie interfejsu użytkownika polega na przeciąganiu komponentów z dostępnej palety, wliczając w to także niewidoczne komponenty, takie jak sensory. W tym widoku można również zmieniać właściwości obiektów, które zostały stworzone. Między innymi istnieje możliwość zmiany położenia, wielkości, układu (pionowy, poziomy).

Designer jest zaprojektowany jako zwykła aplikacja internetowa. Tak więc uruchamia się go, jak zwykłą stronę internetową wpisując jej adres www.

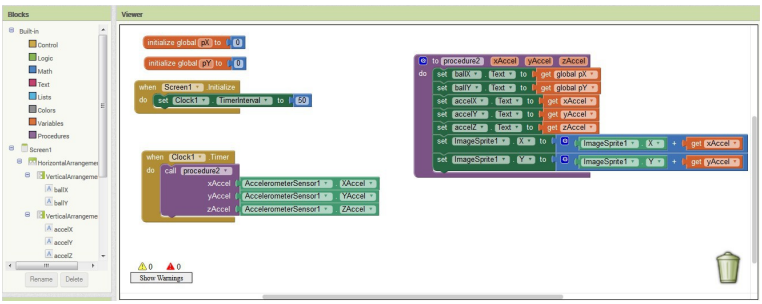
2.2.2 App Inventor Blocks Editor

Drugim widokiem jest Blocks Editor. Zachowanie aplikacji zostaje tutaj zaprogramowane poprzez połączenie odpowiednich bloków. Istnieje możliwość korzystania z bardziej generalnych komponentów, a także z bardziej specyficznych. Dla każdego komponentu, który został stworzony w interfejsie graficznym (Designerze) są dostępne bloki mówiące, co tak naprawdę jest możliwe do zrobienia. Wygląda to w ten sposób, że komponenty są przciągane z dostępnej palety metodą "przeciągnij i upuść", a następnie łączone jak puzzle.



Rysunek 2.1: App Inventor Designer

Ta część aplikacji normalnie reprezentowana jest przez kod napisany przez programistę. Zatem napisanie zachowania aplikacji odbywa się poprzez łączenie puzzli, bez wymogu znajomości języka Java.



Rysunek 2.2: App Inventor Blocks Editor

2.2.3 Android Device Emulator

Android Device Emulator jest to emulator telefonu lub tabletu. Przedstawia on wirtualną wersję smartphonu, w której znajdują się obsługa dotyku ekranu, przyciski systemowe oraz typowe funkcje.

Wprowadzone zmiany, natychmiast reflektują na działanie aplikacji. Nie ma potrzeby jakiejkolwiek kompilacji i uruchamiania aplikacji od nowa. Jeżeli aplikacja zostanie uruchomiana, kompilacja zmienionych fragmentów oraz zainstalowanie ich na emulatorze dzieje się w czasie rzeczywistym. Jest to bardzo wygodna opcja budowania aplikacji i testowania jej. Zmiany, które zrobimy, są od razu widoczne na ekranie.

Rysunek 2.3: Android Device Emulator

2.3 Pozostałe istotne komponenty

W niniejszym rozdziale zawarto pozostałe komponenty, które zostały użyte podczas pisania pracy. Związane są one zarówno z App Inventorem, jak i z aplikacjami napisanymi w Javie.

2.3.1 Android SDK

Adnroid SDK to zestaw narzędzi programistycznych oferowanych dla programistów zamierzających tworzyć aplikacje na platformę Android. Jest on modularny, poprzez SDK Managera możemy zainstalować tylko te komponenty, które nas interesują.

2.3.2 SDK Tools

Android SDK dzieli się na dwie części: SDK Tools oraz Platform Tools. Najważniejsze narzędzia wchodzące w skład pierwszej części to:

- AVD Manager - odpowiedzialny za zarządzanie wirtualnymi urządzeniami z systemem operacyjnym Android. Jest to najłatwiejsza i najwygodniejsza opcja stworzenia nowego wirtualnego urządzenia i odpowiedniego sparametryzowania go.
- SDK Manager - wspomniany wyżej, odpowiedzialny za instalację modułów, które nas interesują.
- Emulator - emualtor systemu android, stworzony przez AVD Managera.
- Dalvik Debug Monitor (DDMS) - jest to narzędzie pomocne w debugowaniu aplikacji. Dostarcza on takich funkcji jak przekierowanie portów, przechwyt obrazu na urządzeniu, informacje o wątkach, stosie, a także o metodach, które są uruchomione jeżeli włączymy ich profilowanie.

2.3.3 Platform Tools

- Android Debug Bridge - narzędzie pozwalające na komunikację z podłączonym urządzeniem. Jest także używany do instalacji i uruchamiania aplikacji. Składa się z 2 części, klienta i serwera, które komunikują się ze sobą.

2.3.4 Apktool

Jest to narzędzie do tak zwanej inżynierii odwrotnej (ang. *Reverse engineering*). Umożliwia ono dekodowanie programu do prawie oryginalnej formy. Następnie, po dokonaniu pewnych modyfikacji, umożliwia ono zbudowanie aplikacji z powrotem do wyjściowej formy.[6]

2.3.5 Keystore

Jest to repozytorium przechowujące certyfikaty bezpieczeństwa. Do zarządzania certyfikatami istnieje narzędzie o nazwie keytool. Umożliwia ono użytkownikom zarządzanie prywatnymi/publicznymi kluczami, certyfikatami jak i podpisem elektornicznym.[7]

2.3.6 Jarsigner

System Android wymaga, aby aplikacje na nim instalowane były cyfrowo podpisane. Dzięki temu system może zweryfikować autora aplikacji. Podpisywanie aplikacji dzielimy na 2 sposoby: tryb debugowania (ang. *Debug mode*)oraz tryb wydania (ang. *Release mode*). Przy korzystaniu ze zintegrowanego środowiska programistycznego zwykle aplikacja zostaje cyfrowo podpisana automatycznie, podczas instalacji jej na telefonie. Jarsigner umożliwia popisanie aplikacji manualnie, korzystając z linii poleceń.

2.3.7 Zużycie procesora

Czas pracy procesora jest to czas, w którym procesor (ang. *CPU*) został użyty to przetwarzania zadanych instrukcji, w przeciwieństwie do oczekiwania na wejście/wyjście lub przejścia w stan oczekiwania (ang. *Idle mode*). Zużycie procesora natomiast mierzone jest w procentach jako całkowita wydajność procesora. Główne zastosowanie to określenie ogólnej zajętości systemu. Wysokie zużycie procesora oznacza zbyt małą moc procesora, lub zbyt wygórowane oczekiwania użytkownika.

2.4 Ważne koncepcje i pojęcia dotyczące App Inventora

Istnieje kilka pojęć, które są warte uwagi. Zostały one zawarte w tym rozdziale.

2.4.1 Publikacja aplikacji na Google Play

Aplikacje zbudowane za pomocą App Inventora mogą zostać przesłane do marketu Google Play. Każda aplikacja, która ma zostać opublikowana musi posiadać wersję kodu (ang. *VersionCode*) oraz nazwę wersji (ang. *VersionName*). Te parametry można ustawić we właściwościach głównego komponentu[9]. Wersja kodu jest to całkowita wartość, która nie jest widoczna dla użytkowników w Google Play. Potrzebna jest do sprawdzenia, czy aplikacja została aktualizowana lub dezaktualizowana do poprzedniej wersji. Nazwa wersji może być dowolna, jednak wg konwencji powinna to być liczba zmiennoprzecinkowa. Domyślnie posiada wartość 1.0. Jest ona zwiększana o 0.1 lub 1 dla małej i dużej zmiany.

Skończony projekt możemy wyeksportować do pliku .apk, który jest automatycznie cyfrowo podpisany kluczem prywatnym powiązanym z naszym kontem. Kiedy tworzymy nową wersję, ten sam klucz jest używany do podpisu. Kiedy urządzenie z system android posiada zainstalowaną aplikację, zapamiętuje on klucz który użyto do podpisu. Celem zainstalowania wyższej wersji należy zastosować do podpisu ten sam klucz.

Repozytorium keystore, w którym znajduje się klucz, domyślnie jest stworzone na serwerze, więc nie potrzebujemy specjalnie go tworzyć. Istnieje również opcja eksportu i importu repozytorium. Jest ona przydatna przy przenoszeniu projektu na inny serwer.

2.4.2 Paleta z dostępnymi komponentami

- **User Interface** - w większości widoczne komponenty, które są związane z interfejsem użytkownika.
  - Button - przycisk.
  - CheckBox - pole wyboru.
  - DatePicker - komponent dający możliwość wyboru daty.
  - Image - komponent umożliwiający wyświetlenie przesłanego zdjęcia.
  - Label - etykieta, na której zwykle wyświetlany jest kawałek tekstu.
  - ListPicker - komponent, który po kliknięciu wyświetla listę, z której użytkownik może wybrać wartość. Daje on także możliwość automatycznego osadzenia wyszukiwarki na liście.
  - ListView - komponent, który pozwala na osadzenie i wyświetlenie listy elementów.
  - Notifier - komponent wyświetlający powiadomienia, a także umożliwiający logowanie na 3 poziomach (Error, Warn, Info).
  - TextBox - komponent umożliwiający wpisywanie tekstu.
  - PasswordTextBox - taki sam komponent jak TextBox jednak wpisywany tekst nie jest widoczny dla użytkownika.
  - Slider - jest to pasek postępu, który dodatkowo umożliwia użytkownikowi przeciąganie.
  - Spinner - element wyświetlający pop-up z listą elementów do wyboru.
  - TimePicker - element pozwalający na wybór czasu.
  - WebViewer - komponenet umożliwiający umieszczenie dowolnej strony internetowej w aplikacji.

- **Layout** - Komponenty odpowiedzialne za rozmieszczenie pozostałych komponentów. Są to kontenery, w które mogą zostać umieszczane inne widoczne komponenty.
  - HorizontalArrangement - elementy umieszczone w tym kontenerze są układanej od lewej do prawej.
  - VerticalArrangement - odwrotne działanie do poprzedniego komponentu - elementy umieszcza się od góry do dołu.
  - TableArrangement - element umożliwiający ustawienie elementów postaci tabularnej
- **Media** - Komponenty związane głównie z dźwiękiem oraz kamerą.
  - Camcorder - komponent umożliwiający nagrywanie filmów. Istnieje możliwość nadania nazwy pliku zawierającego nagranie.
  - Camera - komponent umożliwiający robienie zdjęć i zapisywanie ich.
  - ImagePicker - komponent uruchamiający galerię zdjęć zawartą na telefonie i dający możliwość wyboru zdjęciu. Zdjęcie, po wybraniu, jest kopiowane na kartę SD (maksymalna ilość zdjęć to 10). Następnie możemy z danego zdjęcia skorzystać w aplikacji i je wyświetlić.
  - Player - komponent odtwarzający muzykę, a także odpowiedzialny za wywołanie wibracji w telefonie.
  - Sound - komponent odtwarzający dźwięki, w porównaniu do poprzedniego, dźwięki powinny mieć krótki czas trwania, podczas gdy muzyka może być odtwarzana stosunkowo długo.
  - SoundRecorder - komponent nagrywający dźwięk.
  - SpeechRecognizer - komponent umożliwiający rozpoznanie mowy i stworzenie z niej tekstu.
  - TextToSpeech - komponent o odwrotnym działaniu do poprzedniego, zamieniający tekst na mowę. Wsparcie dla języków: czeskiego, hiszpańskiego, niemieckiego, francuskiego, duńskiego, włoskiego, polskiego, angielskiego.
  - VideoPlayer - komponent umożliwiający odtwarzanie filmu podczas działającej aplikacji. Pliki wideo muszą mieć poniżej 1MB, dodatkowo rozmiar całkowitej aplikacji wynosi maksymalnie 5MB.
  - YandexTranslate - komponent umożliwiający tłumaczenie tekstu pomiędzy językami. Korzysta on z serwisu o nazwie Yandex - https://translate.yandex.com. Dodatkowo urządzenie musi być podłączone do Internetu.
- **Drawing and Animation** Komponenty umożliwiające rysowanie oraz animacje.
  - Canvas - płótno, na którym możemy rysować dwuwymiarowe obrazki (ang. *Sprite*). Obrazki te mogą się na płótnie poruszać. Każda lokalizacja na płótnie jest specyfikowana za pomocą współrzędnych X,Y.
  - ImageSprite - obrazek, który możemy umieścić na płótnie i który może reagować na dotyk, przeciąganie.
  - Ball - jest to ImageSprite, który ma ustawiony obrazek jako koło o określanym kolorze.
- **Sensors** Niektóre z sensorów, dostępnych na telefonie. Wszystkie z tych komponentów są niewidoczne.
  - AccelerometerSenser - akcelerometr, komponent, który umożliwia wykrycie trzęsienia telefonem, podaje wartości, odpowiadające aktualnemu wychyleniu telefonu.
  - BarcodeScanner - komponent umożliwiający skanowanie kodów kreskowych, jednak musimy posiadać dodatkowo aplikację do tego zainstalowaną już na telefonie.
  - Clock - Zegar oraz czasomierz
  - LocationSensor - komponent dostarczający informacje o położeniu gdzie się znajdujemy, czyli szerokość i długość geograficzną. Informacje te mogą nie być od razu dostępne i musimy na nie poczekać.

  - NearField - komponent oferujący możliwośći NFC. Dotychczas komponent ten umożliwia czytanie i wysyłanie tagów tekstowych.
  - OrientationSensor - żyroskop - komponent dostarczający informację o urządzeniu w 3 wymiarach.

- **Social** - komponenty związane z kontaktami, e-mailami i serwisami społecznościowymi.
  - ContactPicker - przycisk, którego naciśnięcie powoduje wyświetlenie podlegających wyborowi kontaktów społecznościowych. Po dokonaniu wyboru użytkownik zyskuje dostęp do następujących danych: nazwa, e-maile, telefony, zdjęcie kontaktu.
  - EmailPicker - Textbox oferujący użytkownikowi pomoc, która polega na (automatycznym/ natychmiastowym) wyświetleniu listy adresów elektronicznych pasujących do aktualnie wpisywanego tekstu.
  - PhoneCall - komponent umożliwiający uruchomienie funkcji dzownienia do osoby, którą wcześniej ustawimy jako właściwość komponentu.
  - PhoneNumberPicker - przycisk o podobnym działaniu do komponentu ContactPicker.
  - Sharing - niewidoczny komponent, który umożliwia udostępnienie wiadomości lub pliku innym aplikacjom.
  - Texting - komponent odpowiedzialny za zarządzanie wiadomościami. Jeżeli aplikacja działa w tle, zdarzenie przyjścia wiadomości również będzie uruchomione. Nawet jeżeli aplikacja nie jest uruchomiona pojawi się powiadomienie o przyjściu wiadomości, po kliknięciu w nie uruchomi się aplikacja.
  - Twitter - komponent umożliwiający komunikację z serwisem internetowym Twitter. Jeżeli użytkownik zostanie pozytywnie zautentykowany i zautoryzowany, pojawia się wiele możliwości, m.in. szukanie tweetów, wysyłanie tweetów, wiadomości, obrazków.

- **Storage** - komponenty odpowiedzialne za przechowywanie danych.
  - File - niewidoczny komponent umożliwiający zapis i odczyt pliku. Domyślne ustawienia zapisują pliki do prywatnego katalogu App Inventora, jednak istnieje możliwość ustawienia innej ścieżki
  - FusiontablesControl - komponent, który komunikuje się z serwisem internetowym dostarczonym przez Google o nazwie Fusion Tables. Tabele te umożliwiają wizualizację, udostępnienie, zapis danych. Komponent ten daje możliwość dostępu do tych danych, a także jej edycji.
  - TinyDB - baza danych dla aplikacji. Jest to odpowiednik klasy Javy - SharedPreferences. Można sobie ją wyobrazić jako mapę - klucz/wartość.
  - TinyWebDB - niewidoczny komponent, komunikujący się z internetową bazą danych.

- **Connectivity** - komponenty umożliwiające komunikację i uruchamianie innych apliakcji.
  - ActivityStarter - komponent do uruchamiania zewnętrznych Activity. Przez Activity są rozumiane: inne aplikacje, kamera, wyszukiwarka internetowa, otwieranie strony internetowej, aplikacji mapy w zadanej lokalizacji. Tak naprawdę, możliwe jest wystartowanie dowolnej aplikacji, jednak trzeba znać nazwę pakietu (ang. *package name*) oraz nazwę klasy (ang. *class name*).
  - BluetoothClient - komponent blutetooth klienta.
  - BluetoothServer - komponent bluetooth serwera.
  - Web - komponenet umożliwiający wysyłanie żądań typu REST do serwera. Dostarcza od funkcje: GET, POST, PUT, DELETE.

- **LEGO MINDSTORMS** - komponenty dostarczające kontrolę nad robotami poprzez bluetooth. W niniejszej pracy magisterskiej zostały pominięte, ze względu na brak powyższych robotów.

2.4.3 Dostępne bloki

- Control - bloki odpowiedzialne za przepływ informacji w aplikacji. Znajdują się tutaj instrukcje warunkowe if-else, pętle for, foreach, while, do-while. Inne bloki w tej sekcji są odpowiedzialne za otwieranie innych aplikacji lub zamykanie aktualnej.
- Logic - bloki odpowiadające za logikę. Kiedy istnieje potrzeba stworzenia instrukcji warunkowej zazwyczaj korzysta się z tych bloków. Sprawdzają one, czy zmienne są takie same lub różne. Są tutaj też wartości true i false.
- Math - bloki odpowiedzialne za wyrażenia matematyczne, min. takie jak dodawanie, odejmowanie, mnożenie, dzielenie. Ale są tu również funkcji pomocnicze, np. konwersja radianów do stopni lub odwrotnie, funkcje trygonometryczne, minimum dla zadanych argumentów, losowa wartość.
- Test - bloki odpowiedzialne za zarządzanie tekstem. Jest tutaj większość funkcji znanej klasy String z Javy. Sprawdzenie czy zmienna jest pusta, ilość znaków, zamiana występnień danego elementu.
- Lists - bloki tworzące listy oraz zarządzające nimi. Istnieje możliwość dodawania, usuwania, zamiany elementów na liście za pomocą oferowanych funkcji. Dodatkowe funkcji znajdujące się tutaj to funkcje konwertujące listę do formatu wiersza lub tabeli, który można następnie umieścić w pliku csv.
- Colors - bloki z kolorami, które można przypisać stworzonym komponentom. Istnieje także możliwość zdefiniowania własnego koloru.
- Variables - bloki odpowiedzialne za tworzenie zmiennych lokalnych i globalnych. Znajdują się tutaj też bloki inicjalizujące i odczytujące wartość zmiennych (ang. *settery i gettery*).
- Procedures - bloki tworzące procedury oraz funkcje zwracające wartość.

2.4.4 Android - przechowywanie danych

Urządzenie z systemem Android dostarcza szereg mechanizmów dotyczących przechowywania danych. Każdy z nich stosujemy do innych zadań.

- SharedPreferences - przechowuje pary klucz-wartość, czyli niewielkie ilości danych. Mechanizm wykorzystywany jest przede wszystkim do przechowywania ustawień aplikacji.[Mir]
- Baza danych SQLite – wykorzystywana do przechowywania dużej ilości uporządkowanych danych, które ze względu na swoją ilość wymagają wysokiej wydajności dostępu.[Mir]
- Pliki - w związku z wykorzystaniem bazy SQLite, która nie obsługuje przechowywania plików, dane binarne (zdjęcia, filmy, inne pliki) powinniśmy przechowywać w ich niezmiennej formie, na karcie lub pamięci wbudowanej w urządzenie.[Mir]

Rozdział 3

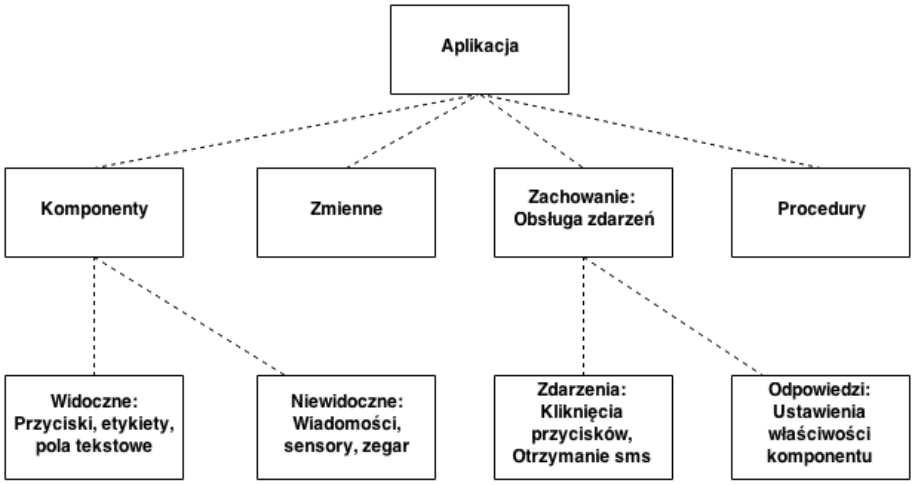
Teoria

3.1 Wstęp

W niniejszym rozdziale zawarto opis architektury oraz główne komponenty wykorzystywane przez App Inventora. Pomoże to zrozumieć zalety oraz wady powyższego narzędzia.

3.2 Architektura

Każda aplikacja ma swoją wewnętrzną strukturę, którą trzeba dokładnie zrozumieć, aby tworzyć efektywne oprogramowanie. Architektura aplikacji składa się głównie z 2 części: komponenty oraz ich zachowanie. Można z pewnym dystansem przyjąć, że za komponenty odpowiada widok Designera, a za zachowanie komponentów widok edytora.



Rysunek 3.1: Architektura aplikacji stworzonej przez App Inventora[3]

3.2.1 Komponenty

Komponenty można podzielić na 2 rodzaje: widoczne oraz niewidoczne.

- Widoczne użytkownik widzi gołym okiem. Należą do nich przyciski, etykiety, pola tekstowe. Definiują one interfejs użytkownika.
- Niewidocznych komponentów, jak sama nazwa wskazuje, użytkownik nie widzi. Nie są one częścią interfejsu. Dostarczają one dostępu do wbudowanych funkcjonalności telefonu. Są to różne sensory np. akcelerometr, moduł gps, komponent zamiany tekstu na mowę itp.

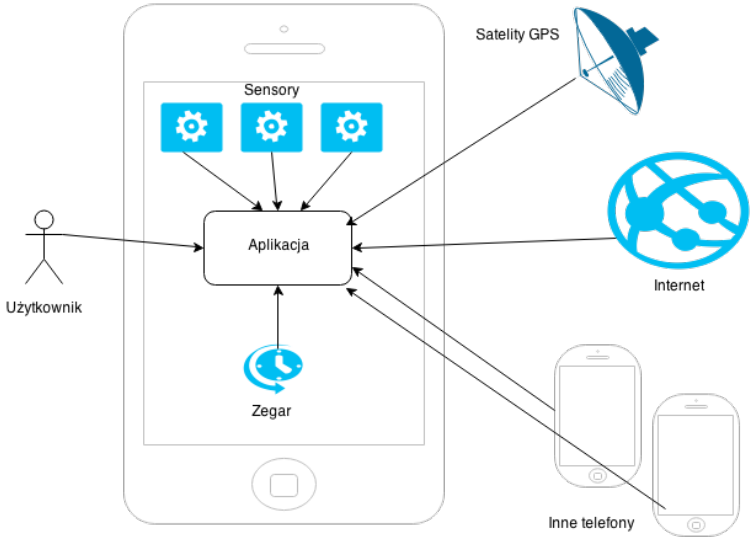
Oba rodzaje komponentów posiadają zbiór swoich właściwości. Właściwości danego komponentu są to informacje jakie komponent posiada. Etykieta posiada między innymi rodzaj, wielkość, dekorację, kolor czcionki, wielkość, widoczność etykiety. Użytkownik nie widzi danych właściwości, obserwuje on rezultat konkretnych ustawień na ekranie urządzenia.

3.2.2 Zachowanie aplikacji

Zrozumienie zasady tworzenia komponentów i ich właściwości jest proste. Nazwy komponentów są intuicyjne, więc nie powinno być problemu z odnalezieniem tego, który interesuje programistę. Z drugiej strony zachowanie komponentów może okazać się bardziej skomplikowane. Mimo wszystko App Inventor stara się wizualizować bloki opisujące zachowanie w jak najprostszej formie.

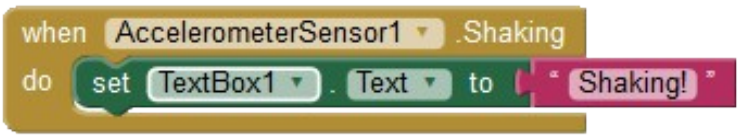
Kiedy zaczynano pisać aplikacje, można było je porównać do recept, gdzie ciąg zdarzeń ukazany jest jako liniowa sekwencja instrukcji. Typowa aplikacja może uruchomić transakcję w banku, dokonać pewnych obliczeń, zmodyfikować stan konta i na koniec wyświetlić nowe saldo.[3]

W dzisiejszych czasach większość aplikacji nie wpasowuje się w powyższy schemat. Zamiast wykonywać ciąg instrukcji w odpowiedniej kolejności, reagują na zdarzenia, które są inicjowane przez użytkownika danej aplikacji. Jednym z przykładów jest kliknięcie przycisku lub trzęsienie telefonem, który jest zaprogramowany tak, aby na każdy bodziec móc umieć odpowiedzieć. Wiele zdarzeń jest inicjowanych przez użytkownika, ale są też wyjątki. Aplikacja może reagować na zdarzenia, które w nie wymagają interakcji z użytkownikiem. Często są to niewidoczne komponenty umieszczone w Designerze. Poniższy rysunek prezentuje aplikację otoczoną wieloma zdarzeniami.



Rysunek 3.2: Aplikacja reagująca na zdarzenia zewnętrzne i wewnętrzne[3]

Jednym z powodów dlaczego App Inventor jest tak intuicyjny jest zastosowanie prostej koncepcji nazywania zdarzeń. Zdefiniowanie zdarzenia polega na przeciągnięciu go z palety na główny ekran, a następnie napisanie konkretnego zachowania. Przykładem takiego zdarzenia jest obsługa akcelerometru. Po zatrzęsieniu telefonem, pojawia się tekst *Shaking!*.



Rysunek 3.3: Obsługa zdarzenia trzęsienia telefonem

Zdarzenia można podzielić w zależności od jego typu:

- Zainicjowane przez użytkownika - najbardziej popularny typ zdarzenia - głównie jest to obsługa zdarzeń dotyku ekranu.
- Inicjalizujące - są wykonywane, gdy dany komponent jest tworzony.
- Czasowe - są uruchamiane, co pewien interwał czasowy.
- Animacje - są zależne od obiektów (spritów) które zostały stworzone. Mogą zostać uruchomione gdy obiekty ze sobą kolidują, wylatują poza ekran.

- Zewnętrzne - są uruchamiane gdy urządzenie odbierze jakiś sygnał zewnętrzny typu, odczyt pozycji urządzenia z satelity, reakcja na przychodzący sms.

Programowanie aplikacji odbywa się poprzez zdefiniowanie interfejsu, a następnie napisanie zachowania danej aplikacji, dla różnych zdarzeń, które mogą wystąpić. Inaczej mówiąc, komponenty tworzone są najpierw w Designerze i tam przypisywane są do nich właściwości. Programista po otrzymaniu interesującego go wyglądu może przystąpić do opisu zdarzeń.

3.3 Debugowanie aplikacji

Najłatwiejszy sposób instalacji i testowania aplikacji odbywa się przez wifi. Musimy pobrać dodatkową aplikację na urządzenie z systemem Android. Następnie na stronie App Inventora uruchamiamy opcję połączenia z telefonem i pojawia nam się na monitorze kod QR, który skanujemy telefonem, z pomocą ściągniętej aplikacji.Po wykonaniu powyższych czynności następuje automatyczna instalacja aplikacji na telefonie. Dzięki aplikacji następuje również automatyczne uaktualnienie wprowadzonych zmian, nie zachodzi konieczność jej ponownego uruchamiania. Powyżej opisany sposób debugowania aplikacji jest rekomendowany, ale należy wspomnieć o istnieniu dwóch innych możliwości. Jedna z nich odnosi się do przypadku, gdy nie posiadamy urządzenia z systemem Android. Możemy wówczas użyć emulatora. Trzecia opcja to możliwość połączenia telefonu z komputerem i aplikacją przez kabel USB.

Rozdział 4

Zastosowane podejście

W niniejszym rozdziale zawarto opis zastosowanego podejścia, do porównania programowania wizualnego i programowania natywnego.

4.1 Wstęp

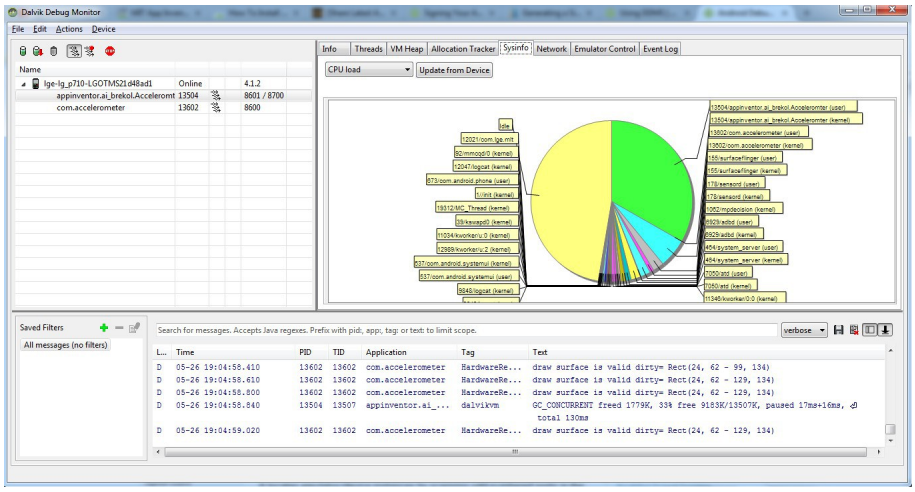
Zastosowane podejście polegało na stworzeniu jak największej ilości aplikacji, wykorzystujących różne komponenty. Następnie należało wykreować analogiczne aplikacje w języku Java. Dysponując obszerną, pokrywającą niemal wszystkie możliwości App Inventora liczbą aplikacji programista jest w stanie udzielić odpowiedzi na wiele pytań dotyczących tego narzędzia, postawionych we wstępie pracy. (1.2)

4.2 Dalvik Debug Monitor

W systemie Android każda aplikacja jest uruchamiana w osobnym procesie, a każdy z procesów działa na swojej własnej wirtualnej maszynie. Każda z tych wirtualnych maszyn wystawia unikalny port, do którego może się podłączyć debbuger. Dalvik Debug Monitor (2.3.2) zaraz po starcie podłącza się do Android Debug Bridge (ADB) - narzędzia, które pozwala na komunikację z podłączonym urządzeniem (2.3.3).

Po podłączeniu urządzenia tworzony jest serwis monitorujący pomiędzy ADB a DDMS, który powiadamia DDMS, kiedy wirtualna maszyna na urządzeniu jest uruchomiona lub zakończona. Gdy wirtualna maszyna wystartuje DDMS odbiera ID (pid) procesu uruchomionego na tej maszynie korzystając z ADB. Następnie tworzone jest połączenie do debbugera maszyny wirtualnej. Po tych operacjach DDMS jest w stanie komuniokować się z maszyną wirtualną, korzystając z dostosowanego protokołu.[4]

Poniżej widać narzędzie Dalvik Debug Monitor. Na telefonie uruchomione są dwa dodatkowe, poza systemowymi, procesy jednocześnie. Po prawej stronie widać wykres obciążenia procesora, poszczególnych procesów.



Rysunek 4.1: Przykładowy zrzut ekranu DDMS

4.3 Zużycie procesora i pamięci

Każda aplikacja powoduje zużycie procesora oraz zajmuje miejsce w pamięci. Do pomiaru tych wielkości użyto Dalvik Debug Monitor.

Aplikację napisaną w Javie możemy konfigurować dowolnie. Między innymi, ustawiając parametr, mamy możliwość debugowania:

```
android:debbugable="true"
```

Jest to ważne, ponieważ aplikacja (plik *.apk) wyeksportowana z App Inventora jest niemożliwa do debugowania. Powyższy parametr ma fałszywą wartość logiczną. Aby to zmienić trzeba aplikację zdekompilować, aby zobaczyć źródła aplikacji i zmienić opcję debugowania. Dekompilacja odbywa się za pomocą darmowego narzędzia apktool.

```
apktool -d aplikacja.apk
```

Po wykonaniu powyższej komendy zostaje tworzony folder z taką samą nazwą jak nazwa aplikacji. Plik AndroidManifest.xml jest już czytelny i możemy zmienić w nim parametr odpowiadający za debugowanie. Po zmianie, aplikację trzeba skompilować ponownie. Trzeba uruchomić poniższą komendę:

```
apktool -b aplikacja
```

Aplikacja została skompilowana ponownie do pliku *.apk. Aby zainstalować ją na urządzeniu należy ją jeszcze cyfrowo podpisać. Generujemy klucz dla aplikacji:

```
keytool -genkey -v -keystore keystore -alias alias_aplikacji -keyalg RSA
-keysize 2048 -validity 20000
```

Następnie podpisujemy aplikację:

```
jarsigner -verbose -keystore keystore aplikacja.apk alias_aplikacji
```

Ostatecznym krokiem jest zainstalowanie aplikacji na telefonie:

```
adb install aplikacja.apk
```

Dzięki tym wszystkim czynnościom maszyna wirtualna uruchamiająca aplikacja uruchomiona na telefonie udostępnia na port umożliwiający debugowanie. Do tego portu podłącza się Dalvik Debug Monitor, z którego możemy odczytać różne statystyki aplikacji i porównać je ze statystykami aplikacji napisanej natywnie w języku Java.

Rozdział 5

Wyniki eksperymentu

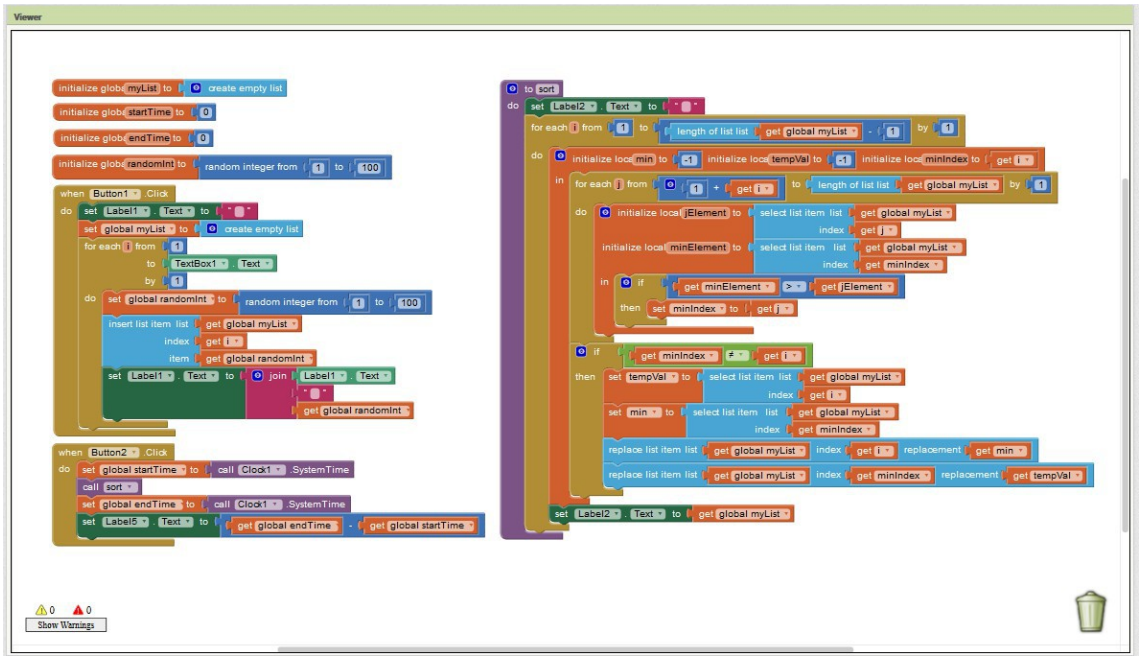
Dany rozdział zawiera wyniki z przeprowadzonych badań oraz wnioski. Każda aplikacja, która została napisana została przestawiona i opisana z różnych perspektyw. Na końcu rozdziału zostały przedstawione wady i zalety obu podejść.

5.1 Stworzone aplikacje

Aplikacje zostały najpierw stworzone w App Inventorze, a następnie zostały przepisane na język Java.

5.1.1 Sortowanie

Aplikacja polega na wygenerowaniu listy losowych elementów, a następnie posortowaniu jej. Do sortowania został użyty prosty algorytm sortowania przez wybieranie ang. *Selection Sort*.



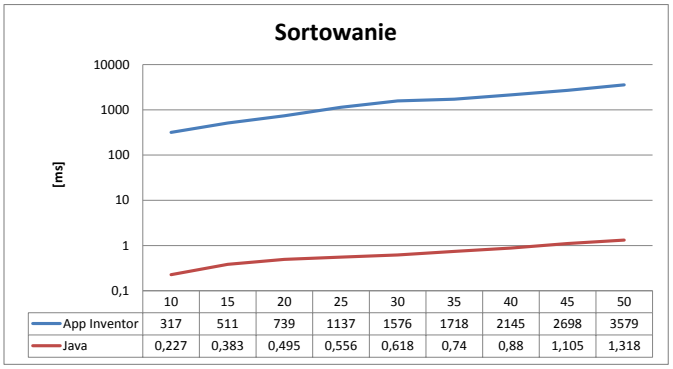
Rysunek 5.1: Aplikacja sortująca - App Inventor

Na powyższym rysunku widać bloki potrzebne do stowrzenia aplikacji w App Inventorze. Bez głębszej analizy zrozumienie działania bloków, może okazać się kłopotliwe. Jest to prosty algorytm, a napisanie go za pomocą dostępnych bloków okazało się skomplikowane. Można sobie łatwo wyobrazić, że napisanie bardziej skomplikowanego algorytmu byłoby bardzo nieczytelne. Ilość użytych bloków zdecydowanie by wzrosła, dodatkowo utrzymanie takiej aplikacji niesie za sobą wysokie koszty wprowadzenia nowych osób do jej rozwijania.

Sortowanie napisanie w Javie jest zrozumiałe dla każdego programisty. Do sortowania została użyta lista, jako odpowiednik listy w App Inventorze, nie ma tam dostępnych tablic.

```
void sort(List<Integer> list){
        for(int i =0;i<list.size()-1;i++){
            int index = i;
            for(int j=i+1;j<list.size();j++){
                if(list.get(j) < list.get(index) ){
                    index = j;
                }
            }
            if(index != i){
                int tmp = list.get(i);
                list.set(i,list.get(index));
                list.set(index,tmp);
            }
        }
    }
```

W algorytmach bardzo ważna jest wydajność. Oba algorytmy działają w ten sam sposób, jednak wydajność sortowania listy napisanej w Javie jest zdecydowanie wyższa. Można to zaobserwować na poniższym wykresie. Przesortowanie bardzo małej liczby elementów zajmuje App Inventorowi bardzo dużo czasu. Przy 25 elementach czas sortowania przekroczył 1 sekundę. Jest to bardzo słaby wynik w porównaniu do sortowania napisanego w Javie. Średnio czas sortowania był 2 tysiące razy mniejszy! Na poniższym uwidaczniającym różnicę wykresie zastosowano skalę logarytmiczną, aby zobaczyć różnicę.



	10	15	20	25	30	35	40	45	50
App Inventor	317	511	739	1137	1576	1718	2145	2698	3579
Java	0,227	0,383	0,495	0,556	0,618	0,74	0,88	1,105	1,318

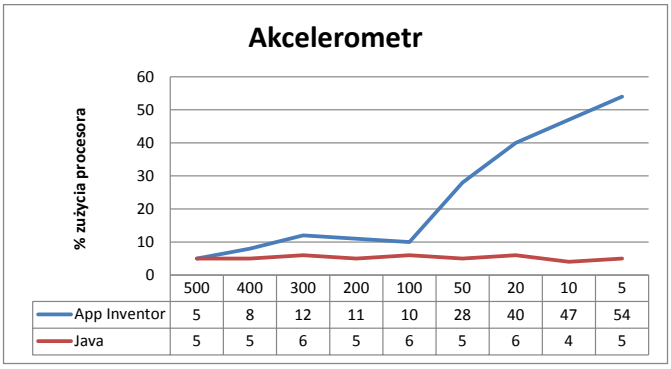
Rysunek 5.2: Wykres przedstawiający czas sortowania

Napisanie omawianej aplikacji w Javie nie było żadnym problemem. Bardzo łatwo było zdebugowano kod i sprawdzono jego poprawność. Stworzenie tej samej aplikacji w App Inventorze nie było trywialne.

5.1.2 Akcelerometr

Kolejną aplikacją jest wykorzystująca akcelerometr. Odczytuje ona dane z akcelerometru, a następnie wyświetla je na ekran telefonu, z zadaną częstotliwością. Na poniższym wykresie przedstawione jest zużycie procesora dla różnych wartości próbkowania. Można zauważyć, że zużycie procesora dla aplikacji napisanej w Javie jest prawie stałe. Dzieje się tak dlatego, że ustawianie częstotliwości próbkowania jest tylko wskazówką dla systemu. Zdarzenia mogą być odbierane szybciej lub wolniej niż zadana częstotliwość. Zazwyczaj są odbierane szybciej. W tym przypadku są odbierane szybciej i zmiana częstotliwości na mniejszą, tak naprawdę nic tutaj nie

zmienia, ponieważ zdarzenia dalej będą odbierane szybciej. Zużycie procesora prawdopodobnie będzie dalej stałe, gdy będziemy zmniejszać częstotliwość.[5]

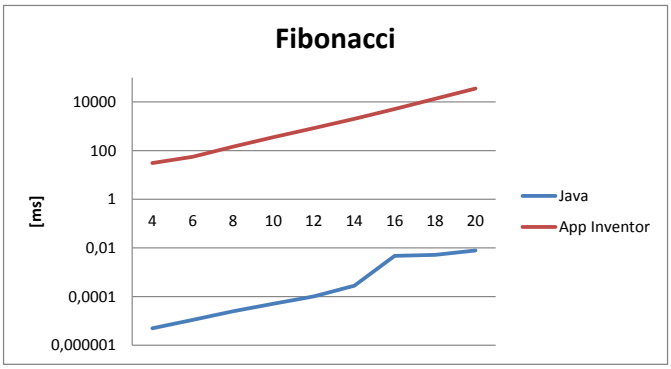


Rysunek 5.3: Wykres przedstawiający zużycie procesora

W AppInventorze nie można zadać bezpośrednio akcelerometrowi częstotliwości próbkowania. Aby to obejść, trzeba dodać nowy komponent Clock, który ma możliwość uruchamiania, co zadany czas. Można podejrzewać, że wydajność akcelerometru jest niezmienna i częstotliwość próbkowania jest stała. Wyraźny spadek wydajności jest przez to, że musimy wywoływać metody w bardzo krótkich odstępach czasu i to, że one dodatkowo odczytują wartość sensora, nie wpływa znacząco na zużycie procesora.

5.1.3 Fibonacci

Następną stworzoną aplikacją jest aplikacja wyliczająca kolejny element ciągu Fibonnaciego. Testuje ona wydajność App Inventora. Złożoność takiego algorytmu to $O(2^n)$, czyli czas wykonywania będzie rósł bardzo szybko. Dodatkowo, kod jest napisany w taki sposób, aby metody wykonywały się rekurencyjnie, jednak aplikacja nie jest w stanie przetestować wielkości stosu. Dla bardzo małych liczb czas wykonywania się algorytmu jest bardzo wysoki.

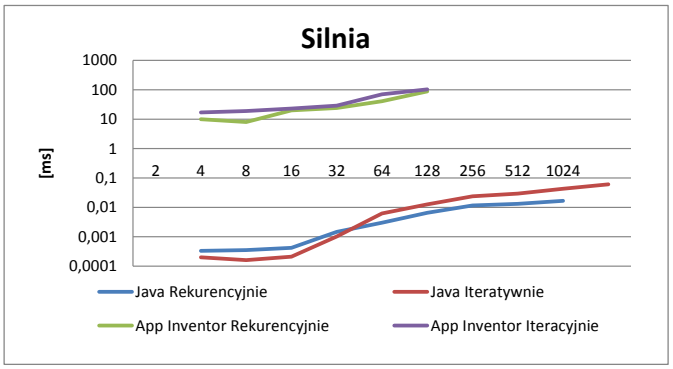


Rysunek 5.4: Wykres przedstawiający czas obliczenia n-tego elementu z ciągu Fibonacciego

Osiągnięty rezultat wydajności nie jest zaskakujący po otrzymaniu wyników z poprzednich programów. Czas obliczenia już początkowych elementów ciągu Fibonnaciego jest bardzo duży. Przy liczeniu dwudziestego elementu aplikacja napisana w App Inventorze potrzebuje ponad pół minuty, podczas gdy, aplikacja napisana w Javie potrzebuje na to około jedną setną sekundy.

5.1.4 Silnia

Następny program oblicza silnię danego elementu. Istnieje możliwość wyboru, iteracyjna lub rekursyjna wersja. Aby zaprezentować oba podejścia na jednym wykresie, zastosowano skalę logarytmiczną.



Rysunek 5.5: Wykres przedstawiający czas obliczenia silni danego elementu

Na powyższym wykresie można zaobserować wiele istotnych elementów. Aplikacja napisana w języku Java nie miała żadnych problemów w podejściu iteracyjnym. Zakres liczb nie został przekroczony, ze względu na możliwość wyboru typu danych. Potrzebna była tutaj klasa dla wielkich liczb całkowitych, dlatego został użyty typ BigInteger. Podczas użycia wersji rekurencyjnej, przy około liczeniu silni dla około 700, program rzuca wyjątek przepełnienia stosu (ang. *Stack overflow*).

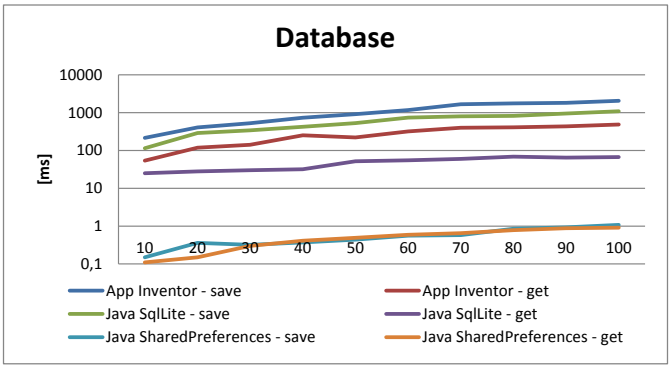
Ciekawa sytuacja występuje dla aplikacji napisanej w App Inventorze. Program rzuca wyjątek podczas działania programu (ang. *Runtime error*). Niestety, nie wiadomo co to za błąd, ponieważ w logach wiadomość o błędzie jest niedostępna. Wiadomość, którą otrzymujemy w logach wygląda następująco:

```
E/com.google.appinventor.components.runtime.util.RuntimeErrorAlert:
No error message available
```

Można się jedynie domyślać, że jest to błąd przepełnienia stosu lub przekroczenia zakresu liczb.

5.1.5 Database

Aplikację stworzono celem kontroli szybkości działania bazy danych oferowanej przez App Inventora. Został tutaj wykorzystany komponent TinyDB. Odpowiada on klasie Javy SharedPreferences, czyli jest to baza danych typu klucz/wartość. Aby odczytać dane z pamięci trzeba znać klucz do tych danych. Jest to bardzo łatwe w przypadku małych ilości danych, jednak trudno jest przechowywać większy struktury, ze względu na potrzebę znania klucza dla każdego wiersza. Duże ilości danych powinny być przechowywane w bazie danych SQLite, ponieważ organizacja danych i zarządzanie nimi jest wydajniejsze. Aby otrzymać część danych z bazy, trzeba posłużyć się językiem zapytań SQL. Daje to możliwość wyszukiwania interesujących nas danych. Z drugiej strony zarządzanie i przeszukiwanie dużych zbiorów danych wpływa na wydajność, więc czytanie danych z bazy danych może być wolniejsze niż czytanie danych z SharedPreferences.

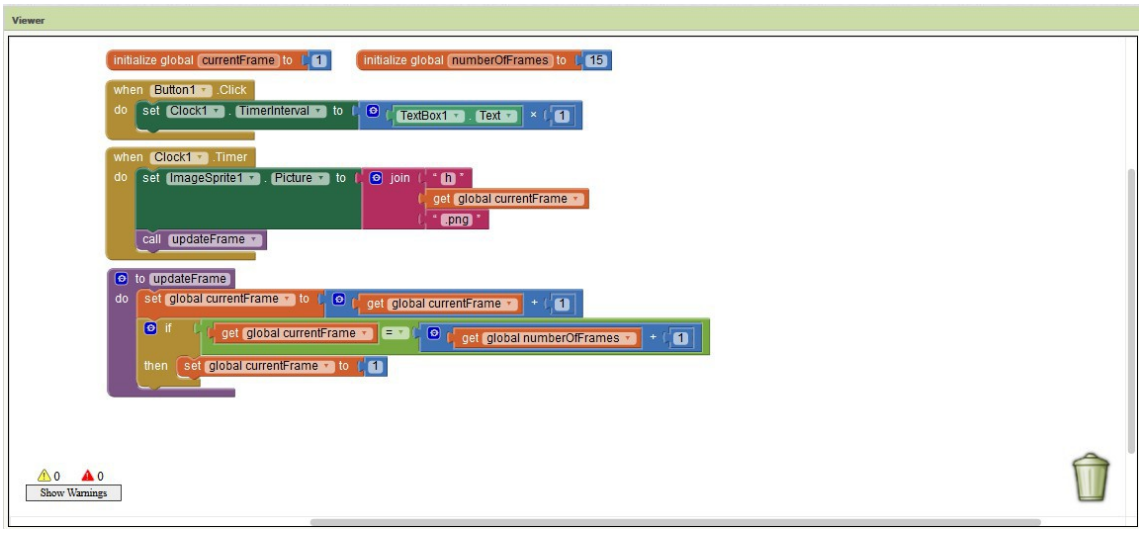


Rysunek 5.6: Wykres przedstawiający czas potrzebny na zapis/odczyt n-elementów

Na powyższym wykresie można zauważyć przewagę wydajności aplikacji napisanej w Javie. App Inventor uzyskał podobny rezultat wydajności, jak aplikacja napisana w Javie, która używa bazy danych SQLite. Jak wspomniano wcześniej odczyt/zapis danych z bazy SQLlite powinien być wolniejszy niż korzystanie z SharedPreferences. SQLite uzykuje tutaj lepszy rezultat TinyDB - odpowiednik SharedPreferences. Można łatwo wyciągnąć wniosek, że wydajność TinyDB jest bardzo niska. Jeżeli porównamy TinyDB i SharedPreferences przewaga aplikacji napisanej w Javie jest ogromna.

5.1.6 Animacja

Aplikacja testuje możliwości animacyjne App Inventora. Wiadomo, że komponenty potrzebne są dostępne, jednak nie wiadomo, czy wydajność pozwoli na płynną animację. Mimo że tworzenie animacji może wydawać się trudne, za pomocą App Inventora stworzenie animacji odbywa się tylko w kilku krokach. Największym problemem było znalezienie odpowiednich klatek, gdzie ostatnia wygląda tak samo jak pierwsza, aby uzyskać możliwość zapętlenia.



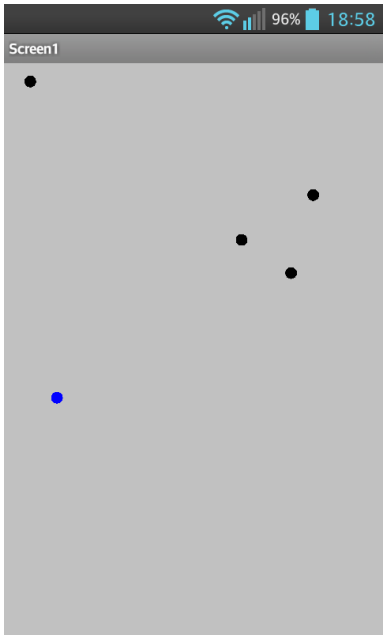
Rysunek 5.7: Bloki potrzebne do stworzenia aplikacji

Na głównym ekranie aplikacji stworzone jest płótno oraz jeden obrazek, który podmieniamy co zadany interwał czasu. Ostatecznym rezulatatem jest płynna animacja. Dopiero przy bardzo

niskim interwale czasowym (szybkiej zmianie klatek), można było odczuć przycinanie się ekranu.

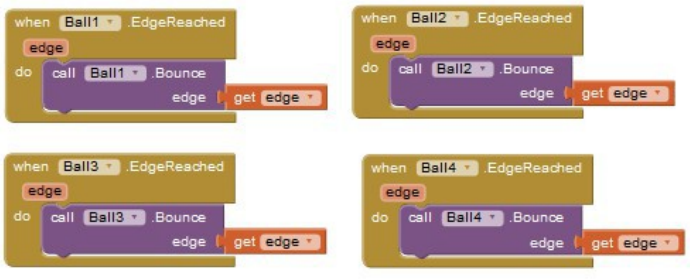
5.1.7 Kolizja elementów

Aplikacja sprawdzająca możliwość detekcji kolizji poruszających się elementów. Na płótnie umieszczono kilka elementów. Elementy w kolorze czarnym poruszają się po płótnie i kiedy dotrą do ściany odbijają się od niej. Element o kolorze niebieskim jest sterowanym za pomocą akcelerometru. Pochylając telefon przesuwamy element po płaszczyźnie.



Rysunek 5.8: Wygląd aplikacji testującej kolizje

Aplikacja działa płynnie, dla takiej ilości elementów. Wadą, którą można tutaj zauważyć, jest brak ogólnego komponentu odpowiedzialnego za zdarzenia lub możliwość przekazania parametru do zdarzenia. Jak widać na poniższym obrazku, tyle ile mamy komponentów, tyle musimy stworzyć zdarzeń odpowiedzialnych za odbicie od ściany. W danym przypadku nie jest to większym problemem. Ale kiedy istnieje potrzeba stworzenia aplikacji, która ma tych komponentów znaczącą ilość, jedną opcją jest wyklikanie zdarzeń po kolei dla wszystkich elementów.



Rysunek 5.9: Bloki odpowiedzialne za zdarzenia kolizji ze ścianą

5.1.8 Activity Starter

Aplikacja daje możliwość wystartowania nowego Activity, w App Inventor istnieje sposobność stworzenia jednej aplikacji, która będzie uruchamiała pozostałe. Odpowiedzialny za to jest komponent ActivityStarter. Wystarczy ustawić mu dwie właściwości, nazwę pakietu oraz klasy. Jeżeli

w dokumentacji aplikacji nie napisano jakie są powyższe nazwy, warto uruchomić aplikację z podłączonym urządzeniem do komputera, aby widzieć logi. Należy wówczas znaleźć wtedy linijkę podobną do poniższej

```
I/ActivityManager:
START {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
flg=0x10200000 cmp=org.mozilla.firefox/.App u=0} from pid 726
```

Jeżeli pojawi się fragmet z parametrem cmp, to nazwą pakietu jest tekst przed ukośnikiem, a nazwą klasy jest cały tekst bez ukośnika. Zatem komponent ActivityStarter daje możliwość uruchomienia prawie każdej aplikacji. Aplikacja nie musi być stworzona w App Inventorze, może to być dowolna aplikacja zainstalowana na urządzeniu. Istnieje również możliwość przekazania dodatkowych parametrów, przy uruchamianiu zewnętrznej aplikacji. Niektóre z nich są nawet zaprojektowane w ten sposób, aby przyjmować dodatkowe parametry podczas uruchamiania. Przykładem są tutaj aplikacja map, która przyjmie jako paramter wartości geograficzne. Innym przykładem jest wyszukiwarka internetowa, która przyjmie jako parametr tekst do wyszukania lub adres do wyświetlenia.

```
Action: android.intent.action.VIEW
DataUri: http://google.pl
```

Ustawiając powyższe parametry, uruchomi się przeglądarka internetowa na stronie http://google.pl. Druga możliwość, jaką daje ActivityStarter, to odbieranie parametrów. W App Inventorze istnieje tylko przekazywanie wartości tekstowych. Aby zwrócić taki parametr trzeba użyć bloku zamykającego aplikację z właściwością do ustawienia (ang. *Close Application With Plain Text*).

5.1.9 Multiple Screens

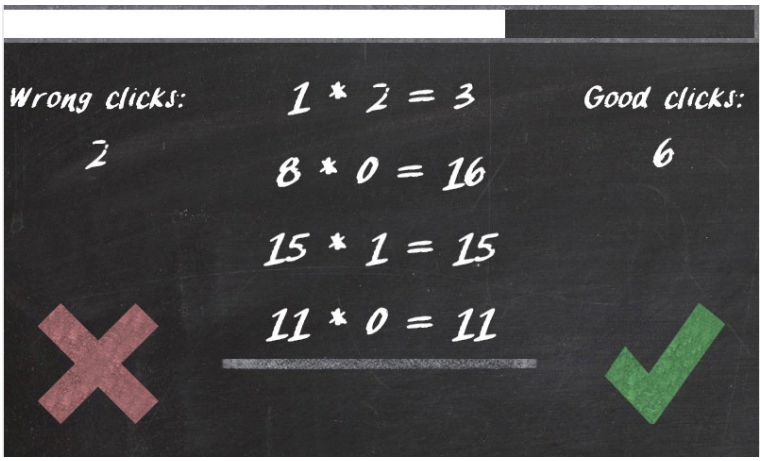
Jeżeli obie aplikacje mają być napisane przez tego samego programistę, warto się zastanowić, czy nie zrobić jednej aplikacji z wieloma ekranami. Zwiększy to użyteczność aplikacji, użytkownicy nie będą musieli instalować dwóch. Używając ActivityStartera dotychczas istnieje możliwość przekazywania tylko parametrów tekstowych. Natomiast przy użyciu wielu ekranów, można przekazywać dodatkowo listy. Kiedy aplikacja korzysta z bazy danych, również jest ona współdzielona pomiędzy ekranami. Otworzony ekran daje możliwość powrotu do ekranu, który go otworzył. Ilość ekranów nie jest ogarniczona, ale zamknięcie ekranu powraca do tego, który go otworzył. Z drugiej strony łatwo to obejść, tworząc ekran, który będzie menadżerem innych ekranów i będzie on uruchamiał pozostałe. Jeżeli z uruchomionego ekranu użytkownik będzie chciał przejść do innego, aplikacja powróci do menadżera z parametrem, a menadżer w odpowiedzi na dany parametr otworzy inny ekran. Zatem tak samo jak w przypadku aplikacji wykorzystującej komponent ActivityStarter istnieje możliwość przekazywania i odbierania parametrów pomiędzy ekranami. Każdy stworzony ekran będzie miał swój wygląd oraz własne komponenty w oknie Designera. Stworzenie jednej bazy danych może być trochę nieintuicyjnie, ponieważ każdy ekran musi posiadać swój komponent TinyDB. Aczkolwiek jest to jedna i ta sama baza danych. Jeżeli jeden ekran zapisze jakąś wartość, drugi może ją od razu odczytać.

5.1.10 Think Faster

App Inventor oferuje bardzo wiele elementów i rozwiązań. Celem sprawdzenia, jak zachowują się one w praktyce podjęto próbę skopiowania gry napisanej wcześniej w języku Java. Jest to gra o nazwie *Think Faster*. Znajduje się ona w markecie Google Play pod poniższym adresem internetowym:

```
https://play.google.com/store/apps/details?id=com.thinkfaster
```

Gra polega na rozwiązywaniu działań matematycznych o różnym stopniu trudności.



Rysunek 5.10: Główny ekran aplikacji

Na powyższym ekranie widać w jaki sposób użytkownik musi rozwiązywać działania matematyczne. Do rozwiązania jest ostatnie działanie na dole ekranu. Można zauważyć, że jest ono błędne, więc gracz powinien kliknąć krzyżyk. Działania będą płynnie przesuwały się z góry na dół. Po nabraniu wprawy, można patrzeć na więcej przykładów, niż tylko to na samym dole i tym samym szybciej je rozwiązywać, co skutkuje większą ilością punktów. Na górze ekranu widać uciekający czas. Po prawidłowym kliknięciu czas zostaje minimalnie zwiększony oraz przez chwilę robi się zielony. Odwrotnie jest przy nieprawidłowym rozwiązaniu przykładu i kliknięciu nie tego przycisku. Część czasu zostaje ucięta i zabarwiona na chwilę na czerwono. Do napisania tej gry użyto dodatkowo bibliotekę AndEngine, która umożliwia animację dwuwymiarową i znacznie ułatwia pisanie kodu.

Rysunek 5.11: Statystyki zebrane przez aplikację Sonar

Można wywnioskować zatem z powyższego opisu oraz ze statystyk przedstawionych przez aplikację Sonar, że *Think Faster* nie jest bardzo skomplikowaną grą. Około dwa tysiące lini kodu, z czego część jest testami, nie robi dużego wrażenia.

Próba skopiowania takiej aplikacji nie zakończyła się całkowitym sukcesem, ale nie można powiedzieć, że zakończyła się porażką.

Na samym początku tworzenia pojawił się problem z paskiem statusu androida, którego nie da się schować w App Inventorze. Mimo wszystko istnieje do tego obejście. Trzeba pobrać plik *apk* na komputer. Następnie go zdekompilować i wyedytować plik *AndroidManifest.xml* dodając następującą właściwość, w każdym Activity, w którym nie ma się pojawiać pasek statusu:

```
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
```

Aplikację następnie trzeba zbudować i podpisać cyfrowo, aby zainstalować ją na urządzeniu.

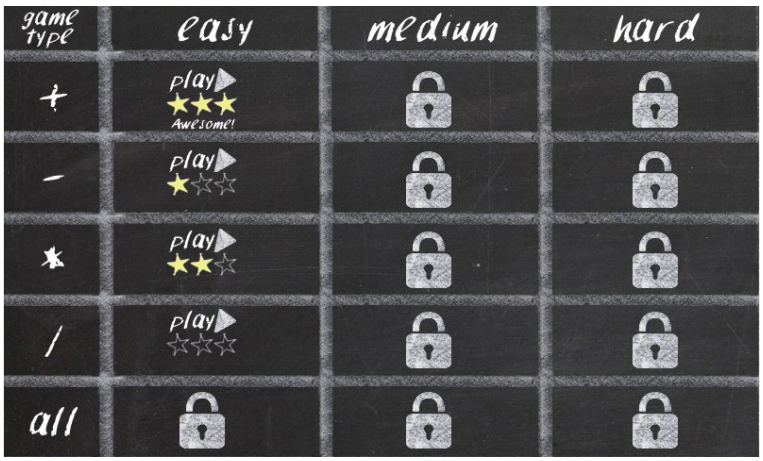
Następnym problemem, jaki powstał, był problem optymalizacyjny. Aplikacja napisana w Javie podczas przełącznia ekranów, ładowała do pamięci grafiki, wyświetlając przy tym napis *Loading....* Jest to niemożliwe do zrealizowania w App Inventorze. Jeżeli ekran posiada wiele elementów, a funkcja inicjalizująca ekran jest skomplikowana i czasochłonna, użytkownik może odnieść wrażenie, że urządzenie przestało odpowiadać. Podczas ładowania ekranów, dodatkowo użytkownikowi wyświetlały się reklamy, które są niemożliwe do stworzenia w App Inventorze.

Ekran menu w oryginalnej aplikacji wygląda następująco:

Rysunek 5.12: Ekran menu gry

Ekran nie jest skomplikowany. Dlatego też całkowicie udało się go odtworzyć w App Inventorze. Wykorzystano komponent Canvas, a poszczególne przyciski były obrazkami z danym napisem. Zaczynając od dołu ekranu, przycisk wyjście udało się obsłużyć i wyjść z aplikacji. Dodatkowo istnieje zdarzenie naciśnięcia klawisza systemowego wstecz, który można obsłużyć we własny sposób. Powiodła się również próba całkowitego odtworzenia ekranu pomocy. Zawiera on jedynie komponent Canvas rozciągnięty na cały ekran, na którym znajduje się obrazek.

Wybierając nową grę użytkownik przechodzi do wyboru ekranu z typem gry. Zaprezentowany ekran widać poniżej:



Rysunek 5.13: Ekran wyboru rodzaju gry

Na tym ekranie użytkownik wybiera poziom trudności oraz przykłady, jakie chciałby rozwiązywać. O ile taki ekran programista jest w stanie stworzyć używając komponentów App Inventora, o tyle wiele trudności przyniosłoby wtedy stworzenie głównego ekranu gry. Poniżej nastąpi uzasadnienie tego stwierdzenia.

Uruchamiając nową grę wiele rzeczy na początku musi zostać stworzone. Pierwszym z elementów jest pasek z uciekającym czasem. Został stworzony jako prostokątny obrazek przesuwający się w lewo. Jedynym problemem było, kiedy nastąpiła kolizja ze krawędzią ekranu. W App Inventorze nie ma możliwości wyłączenia jej. Obejście tego problemu to zmniejszanie rozmiaru obrazka z taką samą prędkością jaką poruszał się on na początku.

Kolejnym elementem są przyciski, mówiące przy działanie matematyczne jest poprawne lub niepoprawne. Przy tworzeniu ich nie pojawiły się żadne problemy. Są to zwykłe obrazki nałożone na płótnie. Posiadają one zdarzenie *TouchUp*, które jest wywoływane w momencie podniesienia palca z przycisku.

Przy tworzeniu następnych elementów zaczynają się trudności. Możliwe jest dynamiczne stworzenie działań matematycznych i umieszczenie ich na etykietach. Jednak niemożliwe jest stworzenie

tych działań i przeniesienie ich na obrazek na płótnie. W celu obejścia takiego problemu można wykorzystać możliwość wcześniejszego, ręcznego stworzenia wymaganych obrazków i załadowanie ich do pamięci telefonu. Aby uzyskać animację znowu pojawia się problem aby dynamicznie stworzyć nowy obrazek na nowy przykład matematyczny i usunąć ten rozwiązany. Rozwiązanie tego problemu to korzystanie tylko z 3-4 przykładów dostępnych na ekranie. Rozwiązanemu przykładowi ustawiamy widoczność na fałsz i przesuwamy do na górę ekranu, podczas gdy reszta elementów przesuwa się standardowo na dół.

Na ekranie są zamieszczone 2 liczniki pokazujące liczbę poprawnych i niepoprawnych kliknięć. Wydawać by się mogło, że są one proste do zaimplementowania. Otóż okazuje się że w App Inventorze bardzo trudno zrobić taki licznik, ponieważ ten licznik jest obrazkiem. Przy ładowaniu ekranu musimy wczytać wszystkie 10 cyfr i po kliknięciu taki nimi manipulować, aby stworzyć odpowiednią liczbę.

Podsumowując powyższe obserwacje należy stwierdzić, że największy problem stanowi dynamiczne ustawianie tekstu na obrazku. Niestety nie jest to możliwe w App Inventorze. Okazuje się jednak, że prawie każdy problem można w mniej lub bardziej pracochłonny sposób rozwiązać. Wniosek jaki można tutaj postawić to, że jeżeli będziemy chcieli tworzyć grę, posiadającą wiele animacji, to lepiej się zastanowić, czy nie skupić się na nauce programowania w Javie i poznaniu bibliotek wspomagających rysowanie grafiki dwuwymiarowej.

5.2 Zalety programowania wizualnego

Główną zaletą programowania wizualnego jest łatwość, z jaką przychodzi pisać aplikacje. Nie trzeba być doświadczonym programistą, aby tworzyć aplikacje za pomocą App Inventora. Jeżeli ktoś jest zainteresowany programowaniem i nie wie jak zacząć, programowanie wizualne może okazać się dobrym wyborem na start. Nauka programowania wizualnego nie jest skomplikowana, wystarczą chęci.

Kolejną zaletą jest nieskomplikowany sposób stworzenia dobrze wyglądającego interfejsu graficznego (ang. *GUI*). Nie znać języku XML aby zdefiniować ładny układ i wygląd ekranu. Wszystkie elementy są przeciągane z palety komponentów, a ich właściwości ustawiane również w prosty sposób.

Następną zaletą są możliwości jakie daje programowanie wizualne. Mimo, że wiele funkcji nie jest dostępnych to aplikacje, które można stworzyć za pomocą App Inventora mogą być bardzo rozbudowane. Komponenty oferowane przez to środowisko pokrywają zdecydowaną większość podstawowych i najbardziej używanych elementów, których się używa podczas pisania aplikacji w języku Java. Jeżeli jakiegoś komponentu brakuje, np. przycisków typu Radio, w internecie jest wiele materiałów w jaki sposób można to obejść.

Myślenie wizualne jest bardziej naturalne dla człowieka. Programowanie w App Inventorze daje programistom możliwość pisania programów poprzez manipulację elementami graficznymi. Dopiero doświadczony programista będzie potrafił sobie wyobrazić na wyższym poziomie abstrakcji kod tekstowy. Dlatego też kod aplikacji Javowej musi spełniać założenia wzorców projektowych, dzięki czemu łatwość nawigowania pomiędzy klasami i metodami jest dużo większa. W App Inventorze stworzenie zadanej funkcjonalności sprowadza się zwykle do użycia pojedynczych bloków, co daje dużą przejrzystość. Dopiero skomplikowane aplkiacje mogłyby rozrosnąć się, tak że zrozumienie ich zajełoby dużo więcej czasu niż takiej samej aplikacji napisanej w języku Java. Jednak jeżeli jest to tak skompilkowana aplikacja warto się zastanowić czy napisanie jej w App Inventorze to dobry pomysł.

App Inventor może być również dobrym narzędziem do tworzenia prototypów. Aplikacje tworzone za pomocą niego powstają bardzo szybko. Stworzony układ graficzny można w krótkim czasie zaprezentować biznesowi, który dzięki temu będzie miał szansę szybko zareagować i skorygować lub zatwierdzić dany interfejs.

5.3 Wady programowania wizualnego

Przynajmniej na razie nie ma możliwości rozszerzenia App Inventora, więc jeżeli istniałaby potrzeba stworzenia lub skorzystania z czegoś, co nie jest wbudowane bezpośrednio w oferowaną platformę, jak np. grafika 3D, to ostatecznie okaże się że projekt nie zostanie zrealizowany.

Implementacja App Inventora nie jest zoptymalizowana dla gier o wysokiej wydajności. Cały framework App Inventora zużywa o wiele więcej mocy procesora, niż programy napisane w języku Java. Przy zwykłym sortowaniu program napisany w Javie jest średnio 2 tysiące razy szybszy.

Stworzenie większego projektu i decyzja o użyciu App Inventora pociąga za sobą zaangażowanie wielu programistów. Nie mogą oni jednak pracować współbieżnie. Współdzielenie projektu odbywa się na zasadzie wyeksportowania go na komputer jako spakowane archiwum. Następnie kolejny programista może go zaimportować. Nie ma to jednak większego sensu, ponieważ kiedy dwie osoby będą pracować nad tym samym projektem, nie będzie można go na końcu scalić. Odwrotnie jest przy pisaniu aplikacji w języku natywnym. Istnieje bardzo wiele narzędzi do rozwiązania tego problemu, są to tzw. systemy zarządzania wersją kodu (ang. *Source code management systems*).

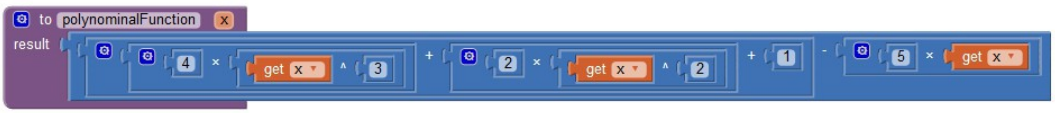
Rozdział 6

Wnioski

App Inventor to sposób pomocy dla kogoś, kto nie miał styczności z programowaniem. Ogólnie rzecz biorąc istnieje kompromis pomiędzy programowaniem wizualnym, a programowaniem natywnym jeżeli chodzi o łatwość użycia, a ekspresywność i moc jaką daje znajomość Javy. Napisanie prostego *Hello World* jest zazwyczaj łatwiejsze w środowiskach oferujących programowanie wizualne. W języku Java, który ma bardzo wiele zastosowań, aby stworzyć i zrozumieć prosty program wyświetlający napis *Hello World* trzeba poznać wiele elementów Javy. Są to między innymi klasy, metody statyczne, pakiety, wywołania metod, różne strumienie do wyświetlenia danego tekstu, składnia języka. Jak widać jest to bardzo dużo elementów.

Jednym z przykładów, który sprawia, że App Inventor jest nastawiony na początkujących programistów jest numerowanie list od 1, zamiast od 0, co może wydawać się nieintuicyjne dla doświadczonych programistów. Głównie to co sprawia że programowanie wizualne jest nastawione na początkujących jest eliminacja możliwości popełnienia błędów składniowych, uniemożliwiając tym samym stworzenie niedziałających programów.

Z drugiej strony stworzenie prostej funkcji wielomianowej: $4x^3 + 2x^2 - 5x + 1$ może okazać się bardziej czasochłonne niż napisanie jej w Javie. Na poniższych rysunkach widać, że funkcja napisana w Javie może okazać się łatwiejsza w zrozumieniu.



Rysunek 6.1: Bloki tworzące powyższą funkcję wielomianową

```
int polynominalFunction(int x){
        return (int)(4*pow(x,3)+2*pow(x,2)-5*x+1);
}
```

Podsumowując można stwierdzić, że niektóre gry, takie jak quizy dadzą się zaimplementować za pomocą App Inventora. Natomiast gdy potrzebujemy płynnej animacji i stojącej za nią bardziej skomplikowanej logiki powinniśmy skorzystać z SDK, które oferuje nam Android. Języki blokowe wydają się być przeznaczone dla początkujących, więc projekty nie są tak zaawansowane, jak te, które są stworzone w języku natywnym. Korzystanie z bloków i przeciąganie ich jest nieporęczne dla złożonych programów.

Język wizualny jest przydatny, jeżeli programista ma na myśli proste projekty. Zawodzą one natomiast, jeżeli chodzi o zarządzanie na wielu poziomach abstrakcji i ziarnistości, jaką wymagają duże projekty. Mają one szansę odnieść sukces jako narzędzie, które będzie wyspecjalizowane w konkretnej domenie. System Android bardzo szybko rozrasta się i często pojawiają się nowe dostępne funkcjonalności. App Inventor jest narzędziem, który potrafi wykorzystać podstawowe, komponenty systemu Android, skupiając się na początkujących programistach. Jego celem nie są doświadczone osoby programujące od wielu lat, ponieważ, będą one chciały skorzystać z funkcji, które nie są dostępne.

Z drugiej strony można zauważyć, że programowanie wizualne jest coraz bardziej popularne. Główne środowiska programistyczne używane przez programistów Javy np. Intellij, Eclipse posiadają graficzne wtyczki umożliwiające tworzenie interfejsu poprzez przeciąganie komponentów. Nie są one idealne, jednak o wiele łatwiej jest z nich skorzystać, a następnie zmienić wygenerowany kod odpowiadający za wygląd ekranu.

Dodatek A

Zawartość płyty DVD

Jako dodatek do tego dokumentu dołączona jest płyta DVD. Zawarte są na niej materiały powiązane z prezentowanym tematem w elektornicznej formie dla potencjalnych osób, które chciałyby kontynuować pracę w typ temacie.

Płyta DVD składa się z następujących elementów:

1. Elektroniczna wersja pracy magisterskiej w formacie PDF jak i w formacie do edycji.
2. Aplikacje stworzone w App Inventorze w plikach o rozszerzeniu .apk.
3. Aplikacje stworzone w Javie odpowiadające aplikacjom App Inventora.

Literatura

[1] App Inventor for Android. [on-line] http://en.wikipedia.org/wiki/App_Inventor_for_Android.

[2] Android: czy to Java, czy nie Java. [on-line] http://gphone.pl/artykuly/android-czy-to-java-czy-nie-java/.

[3] Understanding an App's Architecture. [on-line] http://www.appinventor.org/Architecture2.

[4] Using DDMS. [on-line] http://stuff.mit.edu/afs/sipb/project/android/docs/tools/debugging/ddms.html.

[5] Android - dokumentacja. [on-line] http://developer.android.com/reference/packages.html.

[6] Apktool - dokumentacja. [on-line] https://code.google.com/p/android-apktool.

[7] Keytool - dokumentacja. [on-line] http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html.

[9] App Inventor - Koncepcje. [on-line] http://appinventor.mit.edu/explore/ai2/concepts.html.

[And] Andrew Clark. App Inventor launches second iteration. [on-line] http://newsoffice.mit.edu/2013/app-inventor-launches-second-iteration.

[Mir] Mirosław Stanek. SQLite w Androidzie – kompletny poradnik dla początkujących. [on-line] http://www.android4devs.pl/2011/07/sqlite-androidzie-kompletny-poradnik-dla-poczatkujacych/.

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Jakub Bręk",
    title = "{Programowanie wizualne urządzeń mobilnych}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2014",
}
```