

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

1	Wprowadzenie	1
1.1	Programowanie	1
1.1.1	Programowanie wizualne	1
1.1.2	Programowanie natywne	1
1.2	Cel i zakres pracy magisterskiej	2
1.2.1	Struktura pracy magisterskiej	2
2	Podstawowe pojęcia	3
2.1	App Inventor	3
2.2	Główne komponenty	3
2.2.1	App Inventor Designer	3
2.2.2	App Inventor Blocks Editor	3
2.2.3	Android Device Emulator	4
2.3	Pozostałe istotne komponenty	5
2.3.1	Android SDK	5
2.3.2	SDK Tools	5
2.3.3	Platform Tools	5
2.3.4	Apktool	5
2.3.5	Keystore	5
2.3.6	Jarsigner	5
2.3.7	Zużycie procesora	6
2.4	Ważne koncepcje i pojęcia dotyczące App Inventora	6
2.4.1	Publikacja aplikacji na Google Play	6
2.4.2	Paleta z dostępnymi komponentami	6
2.4.3	Dostępne bloki	9
2.4.4	Android - przechowywanie danych	9
3	Teoria	11
3.1	Wstęp	11
3.2	Architektura	11
3.2.1	Komponenty	11
3.2.2	Zachowanie aplikacji	12
3.3	Debugowanie aplikacji	13
4	Zastosowane podejście	15
4.1	Wstęp	15
4.2	Dalvik Debug Monitor	15
4.3	Zużycie procesora i pamięci	16
4.4	Możliwości App Inventora	16
4.5	Łatwość tworzenia aplikacji	16
5	Wyniki eksperymentu	17
5.1	Stworzone aplikacje	17
5.1.1	Sortowanie	17
5.1.2	Akcelerometr	18
5.1.3	Fibonacci	19
5.1.4	Silnia	20
5.1.5	Database	20

5.1.6	Animacja	21
5.1.7	Kolizja elementów	22
5.1.8	Activity Starter	22
5.1.9	Multiple Screens	23
5.1.10	Think Faster	23
5.2	Zalety programowania wizualnego	25
5.3	Wady programowania wizualnego	26
6	Powiązana praca	27
6.1	Data extraction	27
6.2	Inductive wrappers creation	27
7	Wnioski	29
A	Content of the DVD	31
	Literatura	33

Rozdział 1

Wprowadzenie

1.1 Programowanie

Programowanie jest to między innymi proces tworzenia aplikacji. Jest ono głównie kojarzone z wielkimi ilościami kodu napisanego przez programistów. W większości przypadków tak jest. Jednak sposób pisania kodu może się różnić. Można wydzielić programowanie od bardzo niskiego poziomu do wysokiego poziomu. Przy językach niskiego poziomu, np. Assembler operowanie odbywa się na rejestrach procesora. Operacje są bardzo szybkie, jednak napisanie czegoś bardziej skomplikowanego zajęłoby ogromną ilość lini kodu oraz czasu. Następnie istnieją języki wyższego poziomu, którego składnia ułatwia zrozumienie kodu programu przez osoby, które mają z tym kodem styczność.

Aby jeszcze lepiej zrozumieć pisany kod powstała jeszcze jedna warstwa abstrakcji, gdzie tak naprawdę nie jest wymagana od programistów ani jedna linijka kodu. Jest to programowanie wizualne. Główne źródło tworzonego oprogramowania stanowią bloki graficzne i połączenia między nimi.

1.1.1 Programowanie wizualne

Programowanie wizualne jest to programowanie, które pozwala użytkownikowi tworzyć programy poprzez manipulację elementami graficznymi, inaczej niż w większości przypadków przy użyciu edytorów tekstowych. Prawie wszystkie akcje, które możliwe do osiągnięcia 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" interesujących go obiektach.[1] Są to proste czynności, które nie wymagają głębokiej wiedzy informatycznej. Osoby, które nigdy nie miały do czynienia z programowaniem, nie będą miały większych kłopotów z napisaniem aplikacji.

1.1.2 Programowanie natywne

Programowanie natywne jest to programowanie na daną platformę, a więc napisane oprogramowanie będzie na niej działać bez dodatkowych programów. W przypadku systemu Android jest to język Java. Jest to język obiektowy wysokiego poziomu. Po napisaniu programu, kod jest kompilowany do kodu bajtowego, którym zajmuje się maszyna wirtualna Java (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 Java. 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 perspektyw, 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
- Jak wygląda stworzenie bardziej zaawansowanej aplikacji korzystającej z wielu funkcji telefonu
- Czy jakieś dodatkowe narzędzia są potrzebne do tworzenia aplikacji

Dzięki takiemu porównaniu powstanie czystszy obraz na narzędzie jakim jest App Inventor. Młodsze osoby zainteresowane programowaniem łatwiej będą mogły się zdecydować, w którą stronę pójść. Czy warto w ogóle zawracać sobie głowę App Inventorem, czy od razu uczyć się Javy i mieć dostęp do wszystkich funkcji Androida. Dodatkowo nauczyciele informatyki będą mogli rozważyć naukę podstaw programowania poprzez tworzenie aplikacji na system Android.

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

1.2.1 Struktura pracy magisterskiej

W rozdziale 2 przedstawiono podstawowe pojęcia, które zostały użyte przy pisaniu pracy magisterskiej. Terminy te zostały wyjaśnione, aby bez problemu zrozumieć bardziej skomplikowane zagadnienia.

W rozdziale 2 zawarto teorię dotyczącą App Inventora.

W rozdziale 4 pokazano zastosowane podejście do rozwiązania problemu.

W rozdziale 5 przedstawiono wyniki uzyskane podczas pisania pracy magisterskiej.

Rozdział 2

Podstawowe pojęcia

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

2.1 App Inventor

W grudniu 2013 roku został wydany App Inventor w wersji drugiej. Starsza wersja została nazwana jako Classic. Oba narzędzia są bardzo podobne jednak projekty stworzone w starszej wersji nie mogą zostać zaimportowane do nowszej. W danej pracy magisterskiej skupienie zostało na nowej wersji App Inventora.

App Inventor jest to system, który pozwala na tworzenie aplikacji używając jedynie przeglądarki internetowej. Jest to aplikacja internetowa, umożliwiająca zrobienie programu, przez użytkowników mających bardzo małe pojęcie o programowaniu.

Potencjał aplikacji jest bardzo duży. Można to zauważyć patrząc na ilość aktywnych użytkowników. W maju 2014 roku, liczba aktywnych użytkowników wynosiła 87tys. tygodniowo. Ilość zarejestrowanych to 1,9mln w 195 krajach. Użytkownicy ci stworzyli razem 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

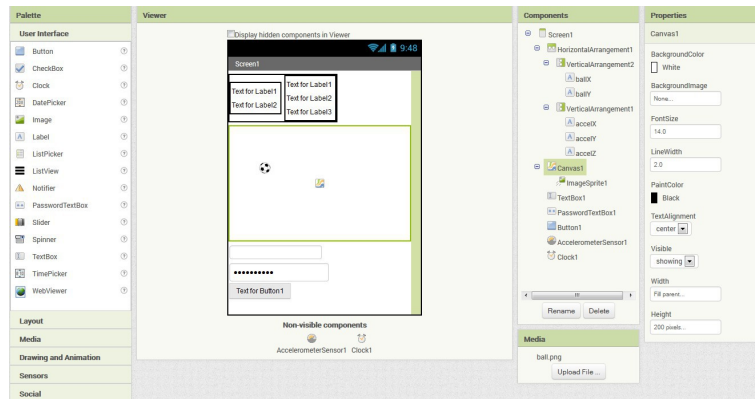
2.2.1 App Inventor Designer

Jednym z głównych widoków jakie 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.

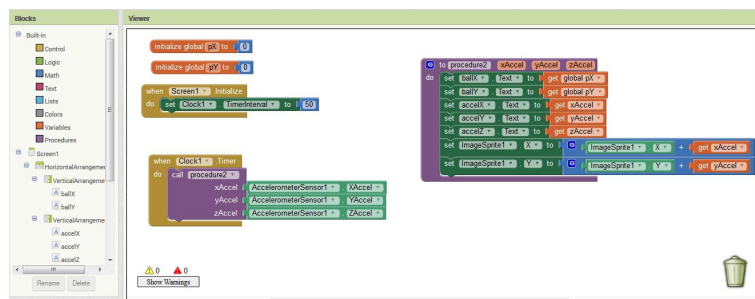
2.2.2 App Inventor Blocks Editor

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 przeciągane 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. Więc napisanie zachowania aplikacji odbywa się poprzez łączenie puzzli, bez 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. Jest to wirtualna wersja smart-phonu, w której znajdują się obsługa dotyku ekranu, przyciski systemowe oraz typowe funkcje.

Zmiany, które zostają wprowadzone, natychmiast reflektują na działanie aplikacji. Nie ma potrzeby jakiegokolwiek kompilacji i uruchamiania aplikacji od nowa. Jeżeli aplikacja zostanie uruchomiona, 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 danym rozdziale zostały zawarte 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

Android SDK - jest to zestaw narzędzi programistycznych, które są oferowane dla programistów, chcących 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 - emulator 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 elektronicznym.[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 całkowitej wydajności 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. *Version-Code*) 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 powiązany 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 został użyty do podpisu. Aby zainstalować wyższą wersję aplikacji ten sam klucz musi zostać użyty do podpisu.

Repozytorium keystore, w którym znajduje się klucz domyślnie jest stworzone na serwerze, więc nie trzeba się tym martwić. 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 - komponent 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 układane od lewej do prawej.
 - VerticalArrangement - odwrotne działanie do poprzedniego komponentu - elementu są umieszczane 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 zdjęcia. 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 trwać 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: czeski, hiszpański, niemiecki, francuski, duński, włoski, polski, angielski.
 - 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 określonym kolorze.
- **Sensors** Niektóre z sensorów, dostępnych na telefonie. Wszystkie z tych komponentów są niewidoczne.
 - AccelerometerSensor - akcelerometr, komponent, który umożliwia wykrycie trzęsienia telefonem, a daje 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 dostarczający możliwości 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, który po naciśnięciu wyświetla listę kontaktów do wyboru. Gdy użytkownik dokona wyboru ma dostęp do następujących danych: nazwa kontaktu, e-maile, telefony, zdjęcie kontaktu.
 - EmailPicker - textBox z pomocą dla użytkownika. Pomoc ta polega na wyświetleniu listy, która zawiera emaile pasujące do wpisywanego tekstu.
 - PhoneCall - komponent umożliwiający uruchomienie dzwonienia 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 przyjscia wiadomości również będzie uruchomione. Nawet jeżeli aplikacja nie jest uruchomiona pojawi się powiadomienie o przyjsciu 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, min. 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 uruchmianie innych aplikacji.
 - ActivityStarter - komponent do uruchamiania zewnętrznych Activity. Przez Activiy 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 (/englishpackage name) oraz nazwę klasy (ang. *class name*).
 - BluetoothClient - komponent bluetooth klienta.
 - BluetoothServer - komponent bluetooth serwera.
 - Web - komponent umożliwiający wysyłanie żądań typu REST do serwera. Dostarcza od funkcje: GET, POST, PUT, DELETE.
- **LEGO MINDSTORMS** - komponenty dostarczające kontrolę nad robotami poprzez bluetooth. W danej pracy magisterskiej zostały pominięte, ze względu na brak powyższych robotów.

2.4.3 Dostępne bloki

- Control - bloki odpowiedzialne za przepływ 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 funkcje pomocniczne, 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 występień danego elementu.
- Lists - bloki tworzące listy oraz zarządzające nimi. Istnieje możliwość dodawania, usuwania, zamiany elementów na liście za pomocą oferowanych funkcji. Dodatkowe funkcje 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. *setter* i *getter*).
- 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 czegoś innego.

- SharedPreferences - przechowuje pary klucz-wartość, czyli niewielkie ilości danych. Mechanizm wykorzystywany jest przede wszystkim do przechowywania ustawień aplikacji.[Mir]
- Baza danych SQLite – wykorzystywane 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 niezmiennnej formie, na karcie lub pamięci wbudowanej w urządzenie.[Mir]

Rozdział 3

Teoria

3.1 Wstęp

W danym rozdziale zostanie zawarty 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 stworzyć 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.



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

3.2.1 Komponenty

Komponenty można podzielić na 2 rodzaje: widoczne oraz niewidoczne.

- Widoczne są to takie, które użytkownik widzi gołym okiem np. 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.

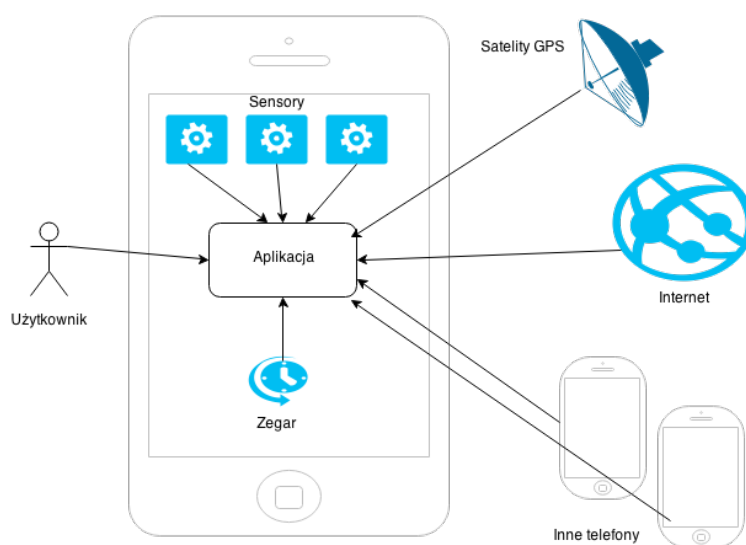
Oba rodzaje komponentów posiadają zbiór swoich właściwości. Właściwości danego komponentu są to informacje jakie komponent posiada. Etykieta posiada między innymi rodzaj, wielkość, dekorację, kolor czcionki, wielkość, widoczność etykiety. Użytkownik nie widzi danych właściwości, obserwuje on rezultat 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 zastosowaniej 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, tworzymy najpierw komponenty w Designerze i ustawiamy im właściwości. Kiedy otrzymaliśmy interesujący nas wygląd zabieramy się za opisanie 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 inventor uruchamiamy opcję połączenia z telefonem i pojawia nam się na monitorze kod QR, który skanujemy telefonem, z pomocą ściągniętej aplikacji. Po tych czynnościach aplikacja zostaje automatycznie zainstalowana na telefonie. Aplikacja dodatkowo automatycznie uaktualnia wprowadzone zmiany, nie trzeba jej uruchamiać ponownie. Jest to rekomendowany sposób, jednak istnieją jeszcze 2 dodatkowe. Jeżeli nie posiadamy urządzenia z systemem Android możemy użyć emulatora. Trzecia opcja to możemy połączyć telefon z komputerem i aplikacją przez kabel USB.

Rozdział 4

Zastosowane podejście

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

4.1 Wstęp

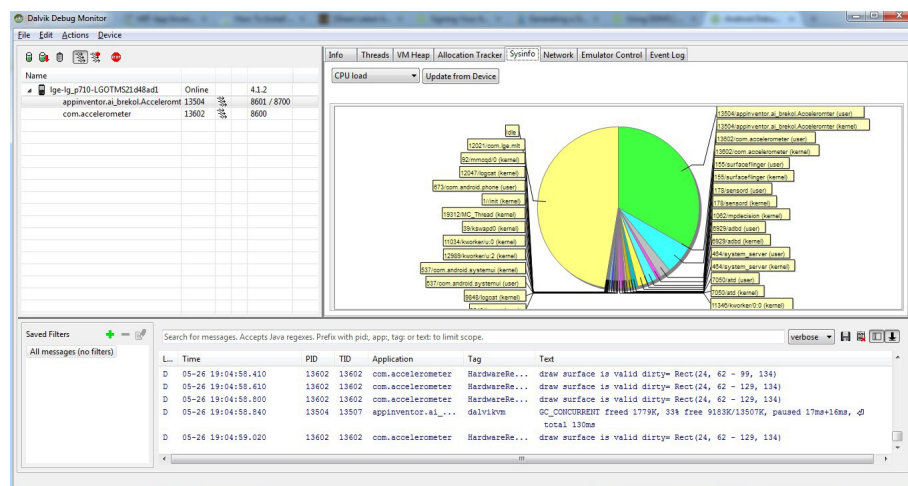
Zastosowane podejście polegało na stworzeniu jak największej ilości aplikacji, wykorzystujących różne komponenty. Następnie stworzenie takich samych aplikacji w języku Java. Mając dużą liczbę aplikacji pokrywającą prawie wszystkie możliwości App Inventora, będziemy mogli odpowiedzieć na wiele pytań jego dotyczących, które zostały postawione 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 został użyty Dalvik Debug Monitor.

Aplikację napisaną w Javie możemy konfigurować dowolnie. Między innymi możemy umożliwić aby była debugowalna, ustawiając parameter:

```
android:debuggable="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 trzeba 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 aplikację 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.

4.4 Możliwości App Inventora

App Inventor posiada bardzo dużą liczbę komponentów, którą możemy użyć. Są jednak pewne braki. Nowe telefony posiadają coraz więcej sensorów oraz innych funkcji, które nie są możliwe do obsłużenia przez powyższe narzędzie.

4.5 Łatwość tworzenia aplikacji

Ilość dokumentacji oraz problemów, z którymi spotkali się użytkownicy może być różna. Stworzenie bardziej zaawansowanej aplikacji może być wyzwaniem dla App Inventora.

Rozdział 5

Wyniki eksperymentu

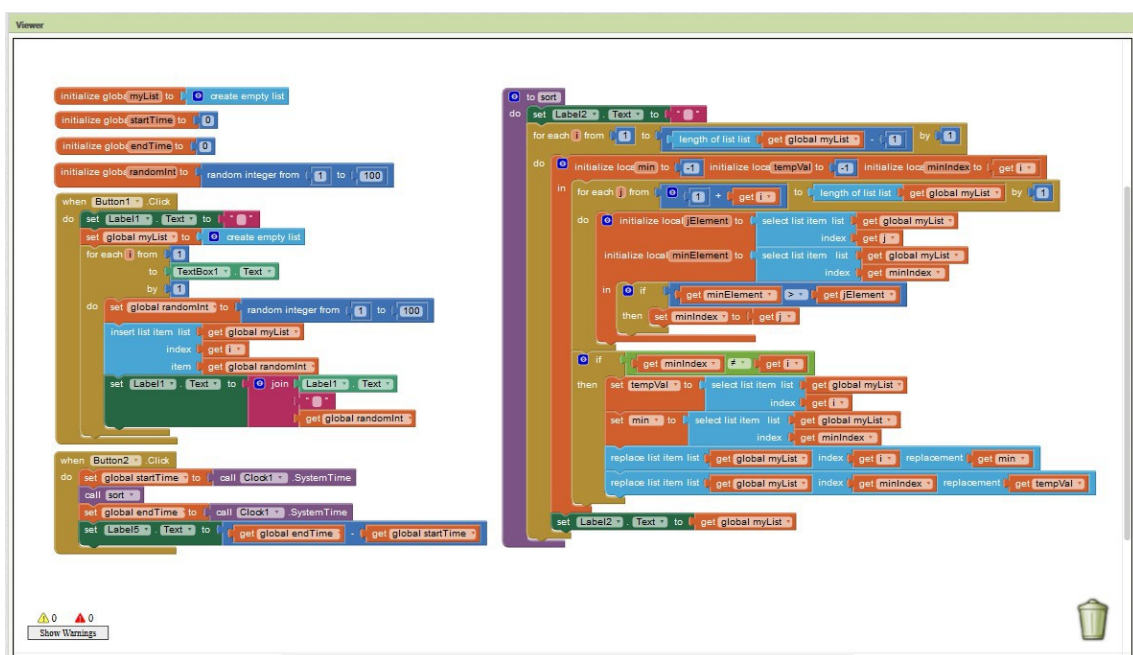
Dany rozdział zawiera wyniki z przeprowadzonych badań oraz wnioski. Każda aplikacja, która została napisana została przedstawiona i opisana z różnych perspektyw. Na końcu rozdziału zostały przedstawione wady i zalety obu podejść.

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*.



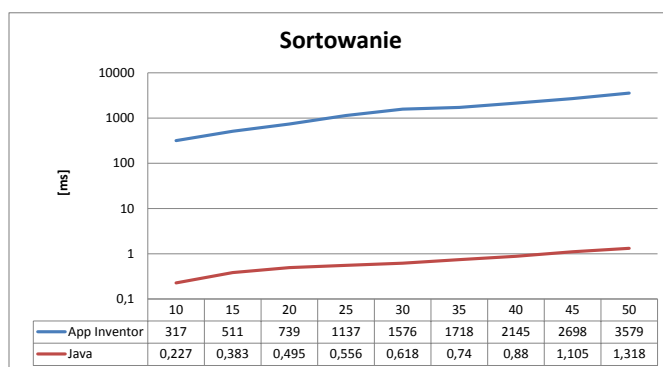
RYSUNEK 5.1: Aplikacja sortująca - App Inventor

Na powyższym rysunku widać bloki potrzebne do stworzenia 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 zdecydowanieby wzrosła, dodatkowo utrzymanie takiej aplikacji niesie za sobą wysokie koszty wprowadzenia nowych osób do jej rozwijania.

Sortowanie napisane 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 danym wykresie została zastosowana skala logarytmiczna, aby zobaczyć różnicę.

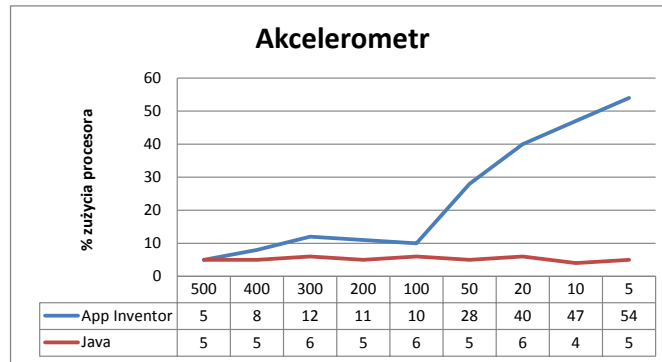


RYSUNEK 5.2: Wykres przedstawiający czas sortowania

Napisanie tej aplikacji w Javie nie było żadnym problemem. Bardzo łatwo było zdebugować kod i sprawdzić 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, 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 to 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]

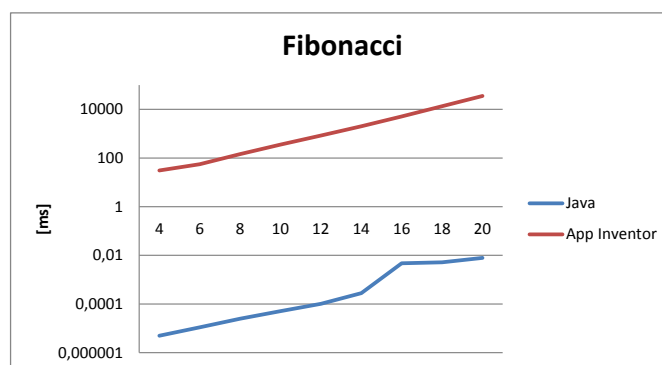


RYSUNEK 5.3: Wykres przedstawiający zużycie procesora

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 tak, 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.



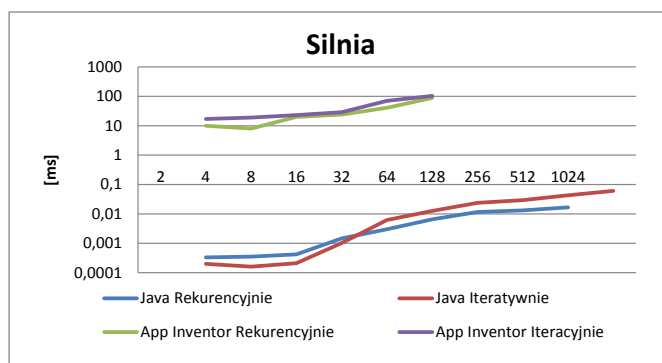
RYSUNEK 5.4: Wykres przedstawiający czas obliczenia n-tego elementu z ciągu Fibonnaciego

Osiągnięty rezultat wydajności nie jest zaskakujący po otrzymaniu wyników z poprzednich programów. Czas obliczenia już początkowych elementów ciągu Fibonnaciego jest bardzo duży.

Przy liczeniu 20 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, została użyta skala logarytmiczna.



RYSUNEK 5.5: Wykres przedstawiający czas obliczenia silni danego elementu

Na powyższym wykresie można zaobserwować 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*).

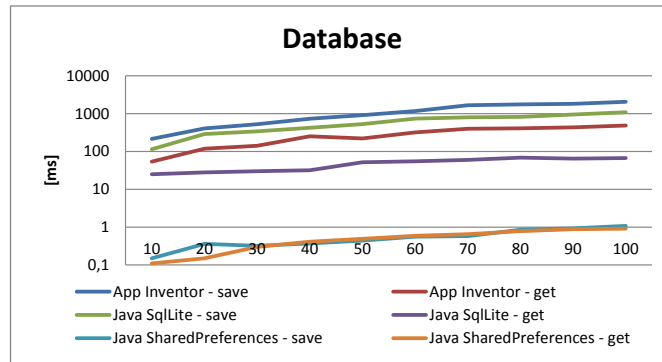
Ciekawa sytuacja występuje dla aplikacji napisanej w App Inventorze. Program rzuca wyjątek podczas działania programu (ang. *Runtime error*). Niestety, nie wiadomo co to za błąd, ponieważ w logach wiadomość o błędzie jest niedostępna. Wiadomość, którą otrzymujemy w logach wygląda następująco:

```
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

Aplikacja stworzona aby sprawdzić szybkość 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 większe 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`.

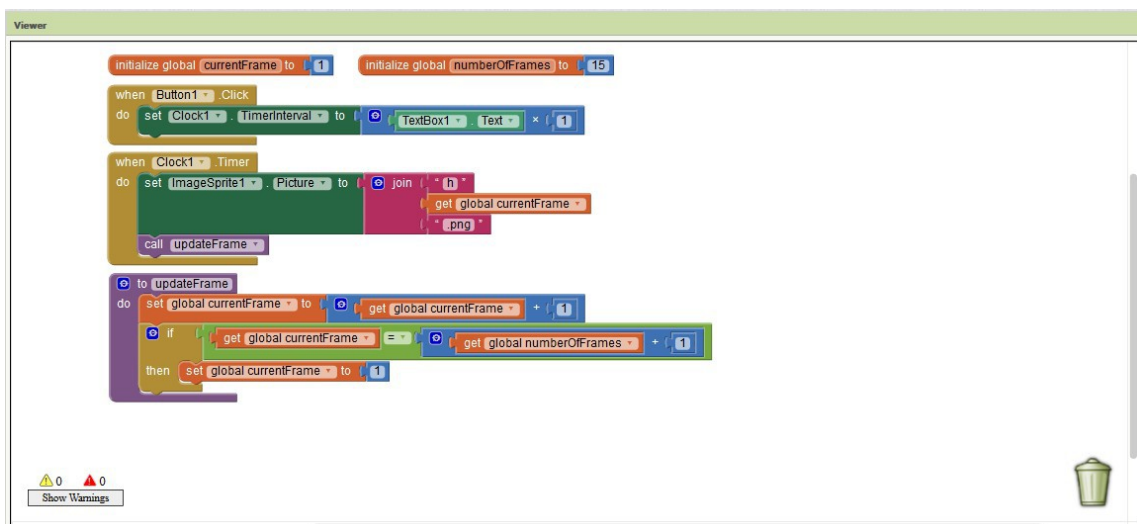


RYSUNEK 5.6: Wykres przedstawiający czas potrzebny na zapis/odczyt n-elementów

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 SQLite powinien być wolniejszy niż korzystanie z SharedPreferences. SQLite użykuje 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 była możliwość zapętlenia.



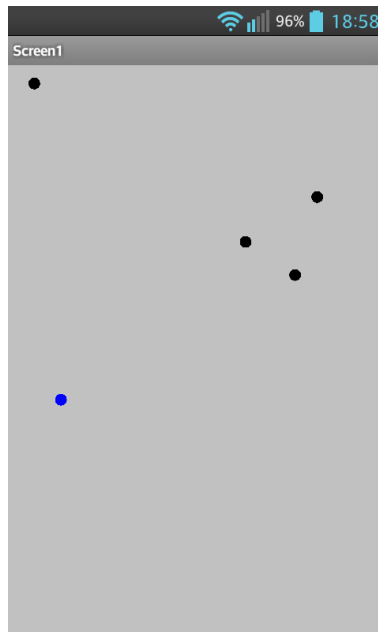
RYSUNEK 5.7: Bloki potrzebne do stworzenia aplikacji

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

niskim interwale czasowym (szybkiej zmianie klatek), można być odczyt przycinanie się ekranu.

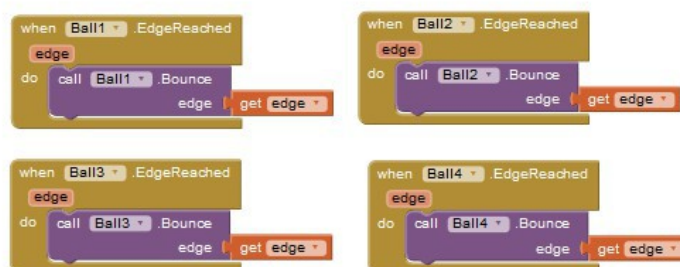
5.1.7 Kolizja elementów

Aplikacja sprawdzająca możliwość detekcji kolzji poruszających się elementów. Na płótnie zostało umieszczone 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 wywołanie 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 2 właściwości, nazwę pakietu oraz klasy. Jeżeli w dokumentacji aplikacji nie jest napisane jakie są powyższe nazwy, warto uruchomić aplikację z

podłączonym urządzeniem do komputera, aby widzieć logi. Trzeba 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 przez 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

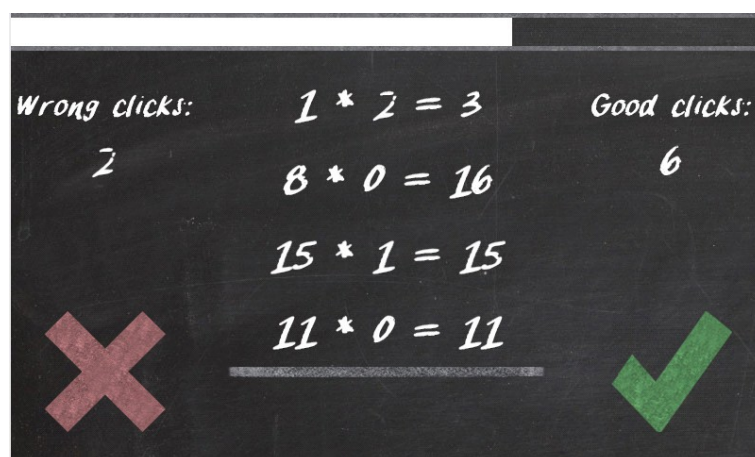
Jeżeli obie aplikacje mają być napisane przez tego samego programistę, warto się zastanowić, czy nie zrobić jednej aplikacji z wieloma ekranami. Zwiększy to użyteczność aplikacji, użytkownicy nie będą musieli instalować dwóch. Używając `ActivityStartera` dotychczas istnieje możliwość przekazywania tylko 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 ograniczona, 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 paramter 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. Aby sprawdzić jak mają 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, które mają różny stopień 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.

Lines Of Code	Files	Functions
2 372	32	198
Java	Directories	Classes
	9	32
	Lines	Statements
	3 172	1 049
		Accessors
		58

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 przełączania ekranów, ładowała do pamięci grafiki, wyświetlając przy tym napis *Loading....* Jest to niemożliwe do zrealizowania w App Inventorze. Jeżeli ekran posiada wiele elementów i 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.

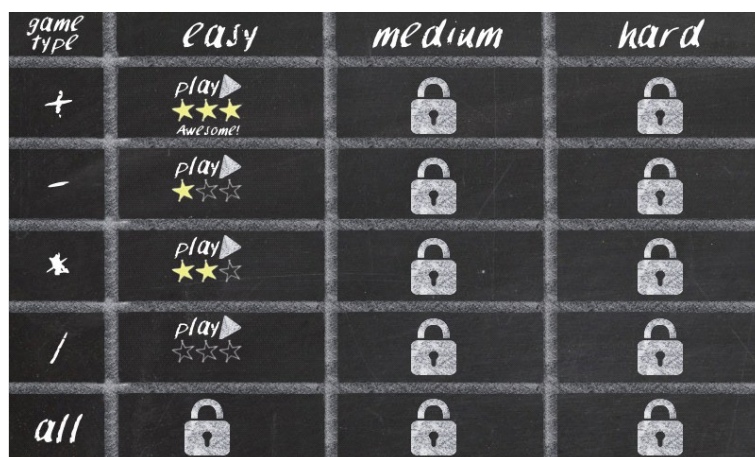
Ekran menu w oryginalnej aplikacji wygląda następująco:



RYSUNEK 5.12: Ekran menu gry

Nie jest on 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. Ekran pomocy również udało się całkowicie odtworzyć. 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 pomoziom 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. W poniższym akapicie jest wytłumaczone dlaczego.

Uruchamiając nową grę wiele rzeczy na początku musi zostać stworzone. Pierwszym elementem jest pasek z uciekającym czasem.

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, wystarczy chęć.

Kolejną zaletą jest łatwość 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 aplikacje mogłyby rozrosnąć się, tak że zrozumienie ich zajęłoby dużo więcej czasu niż takiej samej aplikacji napisanej w języku Java. Jednak jeżeli jest to tak skompilowana aplikacja warto się zastanowić czy napisanie jej w App Inventorze to dobry pomysł.

App Inventor może to 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 tys. razy szybszy.

Stworzenie większego projektu i decyzja o użyciu App Inventora pociąga za sobą zaangażowanie wielu programistów. Nie mogą oni jednak pracować wspólnie. 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 2 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 rozwiązywania tego problemu, są to tzw. systemy zarządzania wersją kodu (ang. *Source code management systems*).

Rozdział 6

Powiązana praca

6.1 Data extraction

6.2 Inductive wrappers creation

Rozdział 7

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 przyszłych 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 7.1: Bloki tworzące powyższą funkcję wielomianową

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

Podsumowując niektóre gry, takie jak quizy mogą być zaimplementowane za pomocą App Inventora, natomiast kiedy 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 jest 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

Content of the DVD

As an addition to this document, the DVD is attached. It provides some materials connected with the presented subject in electronic form for potential users or people, who would want to continue works on this topic.

The DVD content consists of several items:

1. Item 1
2. Item 2
3. Item 3

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/>.



© 2014 Jakub Bręk

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX.

Bib_TE_X:

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