AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PRACA DYPLOMOWA MAGISTERSKA

pt.

„XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX”

Imię i nazwisko dyplomanta: **Paweł Popanda**

Kierunek studiów: **Informatyka Stosowana**

Specjalność:  **Modelowanie i Technologie Informacyjne**

Nr albumu: **226 141**

Promotor: dr inż. Łukasz Rauch

Recenzent: dr Danuta Szeliga

Podpis dyplomanta: Podpis promotora:

Kraków 2013

***(Oświadczenia zgodnie z Regulaminu studiów w AGH):***

***Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.***

Kraków, dnia …… Podpis dyplomanta…………….

**Spis treści**

[1 Wstęp 5](#_Toc393533244)

[2 Analiza problemu 5](#_Toc393533245)

[2.1 Kontekst środowiska mobilnego 6](#_Toc393533246)

[3 Systemy 7](#_Toc393533247)

[3.1 iOS 7](#_Toc393533248)

[3.1.1 Wielozadaniowość w iOS 8](#_Toc393533249)

[3.1.2 Tworzenie i dystrybucja aplikacji na system iOS 9](#_Toc393533250)

[3.1.3 Dostępność środowisk w systemie iOS 10](#_Toc393533251)

[3.2 Android 10](#_Toc393533252)

[3.2.1 Aktualizacje 12](#_Toc393533253)

[3.2.2 Architektura systemu 13](#_Toc393533254)

[3.2.3 Tworzenie i dystrybucja aplikacji Android 16](#_Toc393533255)

[3.3 Windows Phone 16](#_Toc393533256)

[3.3.1 Tworzenie i dystrybucja aplikacji Windows Phone 8 16](#_Toc393533257)

[4 Środowiska międzyplatformowe 16](#_Toc393533258)

[4.1 Unity 16](#_Toc393533259)

[4.2 Web/HTML5/Flash 18](#_Toc393533260)

[4.3 18](#_Toc393533261)

[5 Wymagania przed-implementacyjne 18](#_Toc393533262)

[5.1 Windows Phone 8 19](#_Toc393533263)

[5.1.1 Instalacja Windows 8 w aplikacji Virtualbox 19](#_Toc393533264)

[5.1.2 Instalacja Windows Phone SDK 8 i rejestracja telefonu 22](#_Toc393533265)

[5.1.3 Instalacja oraz konfiguracja środowiska Unity 25](#_Toc393533266)

[6 Zaimplementowane programy testujące 25](#_Toc393533267)

[6.1 Unity: Test grafiki 2D 25](#_Toc393533268)

[6.1.1 Implementacja 26](#_Toc393533269)

[6.1.1.1 Postać gracza 26](#_Toc393533270)

[6.2 Unity: Test ilości obiektów 2D 28](#_Toc393533271)

[7 Wyniki 29](#_Toc393533272)

[8 Podsumowanie i wnioski 29](#_Toc393533273)

[9 Literatura 29](#_Toc393533274)

# Wstęp

# Analiza problemu

W obecnych czasach duża liczba architektur mobilnych oraz różnorodność systemów operacyjnych tychże zaczynają sprawiać problemy osobom tworzącym oprogramowanie. Smartfony posiadają coraz więcej możliwości programowych i sprzętowych, takie jak ekrany wielodotykowe, modemy 4G, WiFi i przede wszystkim możliwości instalowania oprogramowania dostarczonego przez osoby trzecie. Dzięki tym aplikacjom użytkownicy dostali do dyspozycji sporo nowych funkcjonalności, w szczególności wspierających mobilność konsumenta. Dzięki tym aplikacjom przykładowo użytkownik może śledzić swoją pozycję w nieznanym mu mieście, wyświetlać informację w czasie rzeczywistym (rzeczywistość rozszerzona) czy też zapłacić za zakupy w sklepie lub bilet w komunikacji miejskiej. [2] . XXX

Jednak w parze z udogodnieniami dla użytkownika poszły problemy dla twórców tych aplikacji. W świecie mobilnym można wyróżnić cztery podstawowe rodzaje problemów. Pierwszym z nich jest zapewnienie użytkownikowi tego samego doświadczenia z aplikacją niezależnie od platformy, z której korzysta. Interfejs aplikacji powinien być zbliżony pomiędzy platformami i intuicyjny. Drugim problemem jest zużycie zasobów i pobór energii. Pomimo, iż nowoczesne telefony posiadają znacznie więcej mocy niż kiedyś, dalej nie dorównują komputerom klasy PC pod względem dostępnej pamięci czy prędkości procesora. Trzecim z problemów jest utrzymywanie aplikacji. Mobilne systemy operacyjne są często uaktualniane, nierzadko bez wsparcia dla kompatybilności wstecznej. Aplikacje działające na starszych wersjach danego systemu nie muszą działać na nowszych. W jaki więc sposób utrzymywać i testować nowe wersje programu? Ostatnim z problemów jest duża różnorodność sprzętowa oraz systemów operacyjnych pomiędzy producentami współczesnych smartfonów. Każdy z nich korzysta z diametralnie innych technik tworzenia oprogramowania, języków i praktyk.

## Kontekst środowiska mobilnego

Rynek smartfonów dla naszych rozważań możemy uznać za podzielony na dwie grupy: tworzących urządzenia i dostarczających systemy operacyjne. Usługodawcy telefoniczni nie mają znaczenia dla naszych rozmyślań.

Grupa odpowiedzialna za tworzenie urządzeń buduje i komponuje każde urządzenie z konkretnych podzespołów takich jak WiFi, modem 4G, GPS czy akcelerometr, a także konfiguruje telefon i przystosowuje go do korzystania z jednego systemu operacyjnego. Oczywistym jest, że każda z tych firm ma swoje ustalone standardy, zasady i sposoby produkcji urządzeń. Grupa dostarczających systemy operacyjne odpowiedzialna jest za stworzenie połącznia pomiędzy użytkownikiem a leżącym pod spodem sprzętem. Na rynek ten składają się np. Apple z iOS, Open Handset Alliance z Androidem, Microsoft z Windows Phone, RIM z BlackBerry, Samsung z Bada czy też Nokia z Symbianem. Aktualnie dwoma z najpopularniejszych systemów mobilnych są iOS i Android. Firmy te tworzą systemy operacyjne niekompatybilne ze sobą. Niekompatybilność systemu dotyka przeciętnego tworzącego oprogramowanie bardziej niż niekompatybilność sprzętowa, jako że deweloper i tak nie posiada bezpośredniego dostępu do warstwy sprzętowej telefonu, a korzysta z API wystawionego przez tworzących system operacyjny. Poniżej przedstawiono kilka różnic pomiędzy najpopularniejszymi mobilnymi systemami operacyjnymi:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| System operacyjny | Maszyna wirtualna | Język programowania | Interfejs użytkownika | Zarzadzanie pamięcią | IDE | Platformy programistyczne | urządzenia |
| iOS | Brak | Objective-C | Cocoa touch | Liczenie referencji | XCode | Mac Os X | Homogeniczne |
| Android | Dalvik VM | Java | Pliki XML | Garbage Collector | Eclipse | Wiele platform | heterogeniczne |
| Windows Phone 7 | CLR | C# + .NET | Pliki XAML | Garbage Collector | Visual Studio | Windows Vista / 7 | Homogeniczne |
| BlackBerry OS | Java ME | Java | W kodzie | Garbage Collector | Eclipse | Wiele platform | Heterogeniczne |
| Symbian OS | dostępna | C++ | Qt | Ręczne | Qt Creator | Wiele platform | heterogeniczne |

# Systemy

## iOS



System iOS (wcześniej iPhone OS) to system operacyjny firmy Apple Inc. Dystrybuowany z produktami takimi jak iPhone, iPad, iPod Touch, i Apple TV. Po raz pierwszy zaprezentowany w 2007 roku na urządzeniach typu iPhone został rozwinięty i wzbogacony o wsparcie dla urządzeń iPod Touch (wrzesień 2007), iPad (styczeń 2010), iPad Mini (listopad 2012) i dla telewizji drugiej generacji Apple TV (wrzesień 2010). W październiku 2013 roku App Store firmy Apple posiadał ponad milion aplikacji dla iOS, z czego około połowa zoptymalizowana była dla iPada. Aplikacje te zostały ściągnięte ok. 60 miliardów razy. W czwartym kwartale 2012 21% wszystkich sprzedanych urządzeń mobilnych posiadały zainstalowany system iOs. W połowie roku 2012 na świecie znajdowało się około 410 milionów aktywnych urządzeń korzystających z systemu.

Interfejs użytkownika oparty jest o ideę manipulacji bezpośredniej i korzystanie z gestów wielopunktowych. Najczęstszymi sposobami na komunikację z systemem są suwaki, przyciski oraz toglery. System wspiera też rozmaite gesty takie jak ściśnięcie, ściśnięcie odwrotne, dotknięcie i przesunięcie. Wszystkie gesty mają swoje określone definicje w kontekście systemu iOS i jego interfejsu wielo dotykowego. Wbudowane akcelerometry wykorzystywane są w niektórych aplikacjach w odpowiedzi na np. potrząsanie urządzeniem (co często sygnalizuje komendę „cofnij”) lub w celu rozpoznania orientacji urządzenia w przestrzeni trójwymiarowej (np. dla przełączania się pomiędzy sposobami wyświetlania obrazu).

System ma pewne elementy wspólne z systemem OS X takie jak Core Foundation i Foundation, jednak jego interfejsem zajmuje się moduł Cocoa Touch, gdzie w OS X wykorzystywany jest Cocoa. Z powodu tych różnic aplikacje na system OS X nie są kompatybilne z systemem iOS. Dodatkowo, mimo że iOS dzieli z OS X podstawowe cechy systemu Darwin, dostęp do terminala systemowego jest niemożliwy dla użytkownika, przez co system nie jest też w pełni kompatybilny z systemami Unix.

Nowe wersje systemu dystrybuowane są co roku. Najnowsza, iOS 7, została przekazana użytkownikom 18 września 2013 roku. W systemie istnieją cztery abstrakcyjne płaszczyzny: Core OS, Core Services, Media oraz Cocoa Touch.

### Wielozadaniowość w iOS

Wielozadaniowość w iOS miała swój debiut w czerwcu 2010 roku wraz z premierą iOS 4.0. Tylko niektóre urządzenia Apple – iPhone 4, iPhone 3GS i iPod Touch trzeciej generacji – potrafiły korzystać z wielozadaniowości. Sposób implementacji wielozadaniowości w systemie iOS był wielokrotnie krytykowany, za to że aplikacje pracujące w tle zmuszone są korzystać z ograniczonego zasobu funkcji i za wymaganie od deweloperów tworzenia jawnego wsparcia dla tych funkcjonalności w aplikacjach.

Przed premierą systemu iOS 4 wielozadaniowość ograniczona była do wybranych aplikacji instalowanych przez Apple na urządzeniu. Użytkownicy mogli jednak „uwolnić” system („jailbreak”) i uzyskać częściowe, nieoficjalne wsparcie dla wielozadaniowości.

Wraz z iOS 4 wielozadaniowość wspierana jest przez siedem interfejsów (API) działających w tle:

* Audio w tle – aplikacja pozostaje w działaniu w tle dopóki nie zakończy odtwarzania plików multimedialnych
* Voice over IP – aplikacja jest zawieszona jeżeli nie jest aktywne połączenie telefoniczne
* Geolokalizacja w tle – aplikacja informowana jest o zmianie położenia urządzenia
* Powiadomienia push (Push notifications)
* Powiadomienia lokalne – aplikacja planuje powiadomienia, które dostarczane mają być o wybranej porze
* Task completion – aplikacja komunikuje się z systemem operacyjnym w celu ustalenia dodatkowego czasu procesora na zakończenie trwających zadań
* Szybkie zmiany aplikacji – aplikacja nie uruchamia żadnego kodu i może zostać usunięta z pamięci w dowolnym momencie

W systemie iOS 5 zadebiutowały trzy dodatkowe interfejsy:

* Newsstand – aplikacja może pobierać treści w tle tak by były gotowe dla użytkownika w późniejszym czasie
* External accessory – aplikacja komunikuje się i dzieli danymi z urządzeniami zewnętrznymi w ustalonych odstępach czasu
* Bluetooth accessory – aplikacja komunikuje się i dzieli danymi z urządzeniami klasy Bluetooth w ustalonych odstępach czasu

W systemie iOS7 Apple uruchomiło dodatkową funkcjonalność która pozwala wszystkim aplikacjom na dokonywanie aktualizacji w tle. Ta funkcjonalność preferuje aktualizowanie aplikacji z których użytkownik najczęściej korzysta i wszystkie aktualizacje stara się przeprowadzać gdy dostępna jest sieć WiFi, by nie korzystać z przesyłania danych przez moduł telefonii komórkowej.

### Tworzenie i dystrybucja aplikacji na system iOS

Aplikacje muszą być napisane i skompilowane z myślą o systemie iOS i 64 bitowej architekturze lub poprzedniej 32 bitowej. Przeglądarka Safari obsługuje aplikacje sieciowe. Istnieją autoryzowane aplikacje firm trzecich korzystające z kodu natywnego dla urządzeń korzystających z iOS 2.0 i późniejszych.

17 października 2007 roku w liście otwartym opublikowanym nam blogu firmy Apple „Hot news” Steve Jobs ogłosił wydanie pakietu deweloperskiego (SDK) w okolicach lutego 2008 roku[przypissss]. SDK zostało wydane 6 marca 2008 i pozawalało na tworzenie aplikacji dla iPhone, iPod Touch oraz na uruchamianie i testowanie aplikacji w emulatorze. Jednak załadowanie aplikacji na rzeczywiste urządzenie możliwe było tylko (jest nadal?) po opłaceniu składki członkowskiej programu deweloperskiego Apple.

Składki za przynależność do danego programu zostały zatwierdzone na 99 dolarów amerykańskich za każdy z dostępnych programów członkowskich: iOS i OSX. W lipcu 2011 roku Apple rozpowszechniło Xcode w swoim sklepie internetowym do pobrania bez opłat dla wszystkich użytkowników OS X Lion. Od swojego rozpowszechnienia Xcode 3.1 stał się podstawowym środowiskiem dla iOS SDK.

Deweloperzy mogą wybrać dowolną cenę za swoje aplikacje powyżej ustalonego w umowie minimum. Po sprzedaży aplikacji w App Store deweloper zachowuje 70% zysku a pozostałe 30% pozostaje dla Apple. Alternatywnie deweloperowi dana jest możliwość stworzenia aplikacji darmowej – w tym przypadku nie ponosi żadnych kosztów za rozpowszechnianie aplikacji poza wymienioną wcześniej opłatą za przynależność do programu deweloperskiego Apple.

### Dostępność środowisk w systemie iOS

Java, flash, html itp.

## Android



Android jest systemem operacyjnym na urządzenia mobilne opartym o kernel Linuksa rozwijanym przez firmę Google. Interfejs użytkownika oparty jest o manipulację bezpośrednią zaprojektowany jest z myślą o urządzeniach z wbudowanym interfejsem dotykowym takimi jak smartfony i tablety, ale obsługuje też wyspecjalizowane interfejsy telewizyjne (Android TV), zegarki (Android Wear) a nawet samochody (Android Auto). System korzysta z bazy gestów takich jak przewijanie, ściskanie czy też dotknięcie by manipulować obiektami na ekranie a także do obsługi wirtualnej klawiatury.

Od roku 2011 Android posiada największą bazę zainstalowanych aplikacji, a od roku 2013 urządzenia z tym systemem zostały zakupione więcej razy niż urządzenia z systemami Windows Phone, iOS i Mac OS razem wzięte. Od lipca 2013 Google Play (oficjalny sklep z aplikacjami firmy Google) posiadał w swojej bazie ponad milion opublikowanych aplikacji Android i ponad 50 miliardów pobrań tychże. Ankieta wśród deweloperów sporządzona na przełomie kwietnia i maja 2013 roku wykazała, iż 71% deweloperów aplikacji mobilnych tworzy aplikacje na Androida. Na konferencji Google I/O 2014 firma ujawniła, że liczba aktywnych użytkowników (aktywność z ostatnich 30 dni) wzrosła do miliarda z 538 milionów w czerwcu 2013 roku.

Kod źródłowy Androida rozpowszechniany przez Google jest oparty o licencje open source, choć większość urządzeń sprzedawana jest z mieszanką oprogramowania otwartego i komercyjnego. Początkowo rozwijany przez firmę Android, Inc. wspieraną finansowo, a następnie wykupioną przez Google Android zadebiutował na rynku w roku 2007 wraz z założeniem Open Handset Alliance – organizacji dedykowanej rozwijaniu standardów otwartych dla urządzeń mobilnych.

Android jest bardzo popularny wśród firm technologicznych wymagających gotowego, taniego i modularnego systemu operacyjnego dla urządzeń o wysokim skomplikowaniu konstrukcyjnym. Otwarta specyfikacja Androida zaskutkowała sporą społecznością programistów i entuzjastów gotowych na rozwijanie otwartych inicjatyw poprzez dodawanie nowych funkcjonalności do istniejących programów, a nawet do przystosowywania urządzeń, które początkowo korzystały z innych systemów operacyjnych, do Androida.

Android jest rozwijany przez firmę Google do momentu, w którym najnowsze zmiany i aktualizacje są gotowe do rozpowszechnienia, po czym kod źródłowy najnowszej wersji zostaje upubliczniony. Ta wersja kodu bez modyfikacji może zostać uruchomiona tylko na wybranych urządzeniach, zazwyczaj z serii Nexus. Następnie kod źródłowy przechodzi etap adaptacji przez firmy wytwarzające urządzenia. Kod źródłowy Androida nie zawiera komercyjnych sterowników potrzebnych do obsługi niektórych komponentów sprzętowych urządzenia.

### Aktualizacje

Google rozpowszechnia duże, inkrementalne aktualizacje dla Androida co 6-9 miesięcy. Większość aktualizacji możliwa jest do zainstalowania przez bezprzewodowe interfejsy sieciowe. Aktualnie najnowszą wersją systemu jest 4.4 „KitKat”.

W porównaniu do swojego największego rywala – iOS – aktualizacje systemu operacyjnego zazwyczaj docierają do urządzeń końcowych z dużym opóźnieniem. Dla urządzeń które nie należą do serii Nexus aktualizacje często docierają z kilkumiesięcznym opóźnieniem. Problem ten po części spowodowany jest dużym zróżnicowaniem sprzętowym urządzeń z systemem Android, gdzie każda aktualizacja musi być dostosowywana do konkretnego modelu urządzenia. Portowanie aktualizacji jest czasochłonne i wymaga dużych zasobów, dlatego producenci często nadają priorytet nowszym urządzeniom i często pomijają starsze. Z tego powodu starsze urządzenia bardzo często nie otrzymują niektórych aktualizacji jeżeli producent dojdzie do wniosku, że nie warto poświęcać czasu i zasobów na dostosowanie kodu, niezależnie od tego czy starsze urządzenie byłoby w stanie obsługiwać nowsze aktualizacje. Problem ten staje się jeszcze większą przeszkodą, gdy producenci indywidualizują swoje wydanie Androida poprzez dodawanie aplikacji i interfejsów, gdyż te też muszą być dostosowane do każdego nowego wydania. Dodatkowe opóźnienia często wprowadzane są też przez operatorów sieci komórkowych, którzy po otrzymaniu aktualizacji od producenta rozpoczynają swoje własne indywidualizacje a następnie długotrwale testują wewnątrz swojej sieci przed wypuszczeniem poprawki do użytkowników końcowych. W 2012 roku Google rozpoczął proces rozszczepiania pewnych aspektów systemu operacyjnego w celu umożliwienia aktualizacji przez Google Play, bez konieczności aktualizacji całego systemu. Od wprowadzenia tych zmian Google może wprowadzać nowe funkcjonalności do systemu poprzez aktualizację konkretnych aplikacji i serwisów. W wyniku tych zmian Android 4.2 i 4.3 otrzymały znacznie mniej aktualizacji widocznych dla użytkownika, koncentrując się bardziej na małych zmianach i poprawkach dla platform.

### Architektura systemu

Android oparty jest na kernelu Linuksa z linii long-term suport (LTS). Od stycznia 2014 roku aktualne wersje systemu oparte są o kernel w wersji 3.4 lub nowszej, choć konkretna wersja zależy od urządzenia i jego architektury. Kernel Androida posiada dodatkowe zmiany architektury nieobecne w typowym jądrze Linuksa takie jak Binder, ashmem, pmem, logger, wakeclocks oraz inną obsługę w przypadku wyczerpania pamięci (Out of memory exception). Niektóre funkcjonalności opracowane przez Google zostały potem wprowadzone do oficjalnego kernela Linuksa, wśród nich autosleep i wakelocks. Interfejsy pomiędzy jądrami są takie same, acz implementacja Linuksowa pozwala na dwa rodzaje usypiania urządzenia: do pamięci (tradycyjny sposób z którego korzysta Android) oraz do dysku (klasyczna hibernacja znana z komputerów osobistych).

System plików w Androidzie podzielony jest na kilka partycji takich jak /system na sam system i /data na dane użytkownika i jego aplikacje. W przeciwieństwie do dystrybucji Linuksowych użytkownicy Androida nie dostają dostępu root do systemu operacyjnego a delikatne partycje takie jak /system oznaczone są jako tylko do odczytu. Dostęp do użytkownika root jest jednak możliwy poprzez wykorzystanie wad w zabezpieczeniach co często wykorzystywane jest przez społeczność programistów do wprowadzania dodatkowych funkcjonalności do swych urządzeń, ale może też być wykorzystywane do instalacji złośliwego oprogramowania i wirusów.

Na jądrze oparta jest warstwa pośrednia (middleware), biblioteki, API napisane w C oraz aplikacje uruchamiane na frameworku, który wspiera biblioteki kompatybilne z Javą, oparty na Apache Harmony. Android korzysta z wirtualnej maszyny Dalvik z kompilacją just-in-time do uruchamiania plików \*.dex (Dalvik Executable), który zazwyczaj tłumaczony jest z bajt kodu Javy. Android od 4.4 wspiera też eksperymentalną maszynę wirtualną ART, która domyślnie jest wyłączona.

Standardowa biblioteka C Androida – Bionic – została opracowana specjalnie z myślą o Androidzie, jako pochodna standardowej biblioteki C z BSD. Bionic posiada kilka znaczących funkcjonalności opartych o kernel Linuksa i jego rozwijanie odbywa się niezależnie od bazy kodu źródłowego Androida. Głównymi korzyściami z używania Bionic w miejsce biblioteki GNU C (glibc) lub uClibc są inna licencja, mniejsze zapotrzebowanie na zasoby podczas działania programu oraz optymalizacje dla procesorów o niskich częstotliwościach.

W 2012 roku próbując znaleźć bardziej korzystny model licencyjny Google zmienił stos modułu Bluetooth z licencjonowanego przy użyciu GPL BlueZ na moduł z licencją Apache – BlueDroid.

Android nie posiada natywnej obsługi dla systemu X Window, ani nie wspiera pełnego wachlarza funkcjonalności bibliotek GNU. Utrudniło to przenoszenie istniejących aplikacji Linuksowych czy bibliotek do Androida. W wersji piątej Androidowego Native Development Kit (NDK) wprowadzono wsparcie dla aplikacji napisanych całkowicie w C lub C++. Biblioteki napisane w C mogą też być używane w aplikacjach Javowych poprzez wstrzyknięcie i korzystanie z JNI (Java Native Interface).



Rysunek 1. Architektura systemu Android

### Tworzenie i dystrybucja aplikacji Android

## Windows Phone

### Tworzenie i dystrybucja aplikacji Windows Phone 8

Wymagane windows 8+ do developmentu, opisac proces instalowania wirtualnej maszyny + windows 8, a może instalacji VS2012 na windows 7?

+ rejestracja konta na windows Live, unlockowanie urządzenia w celu developmentu (jedno urządzenie za darmo? Reszta platna, sprawdzić ceny) opisac sideloading

# Środowiska międzyplatformowe

## Unity

Unity jest między-platformowym silnikiem gier komputerowych z wbudowanym IDE. Rozwijane jest przez Unity Technologies. Unity pozwala na tworzenie aplikacji 2D i 3D na komputery osobiste, konsole i urządzenia mobilne. Zadebiutowało na Światowej Konferencji Deweloperskiej Apple w 2005 roku. Silnik stworzony został, aby budować projekty na komputerach Mac, jednak szybko zdobył poparcie wystarczające do uzasadnienia stworzenia silnika i narzędzi na inne platformy.



Rysunek 2. Interfejs graficzny edytora Unity

Wersja trzecia silnika Unity została rozpowszechniona we wrześniu 2010 roku i skupiła się na dostarczeniu narzędzi i funkcjonalności, do których przyzwyczajeni byli użytkownicy bardziej rozbudowanych, acz ograniczonych platformą, silników. Pozwoliło to na zdobycie zainteresowania ze strony większych firm deweloperskich jednocześnie dając deweloperom niezależnym dostęp do zaawansowanego narzędzia w przystępnej cenie. Ostatnia duża wersja Unity – Unity 4.0 – została opublikowana w 2012 roku i rozszerzyła silnik o animacje Mecanim i wsparcie dla DirectX 11.

Unity korzysta z następujących mechanizmów w celu renderowania grafiki:

* Direct3D – Windows, Xbox360
* OpenGL – Mac, Windows, Linux
* OpenGL ES – Android, iOS
* Biblioteki komercyjne – konsole

Silnik obsługuje takie funkcjonalności jak: bump mapping, reflection mapping, parallax mapping, SSAO, dynamiczne cienie, renderowanie do tekstury i efekty post-processingowe. Unity pozwala też na importowanie zasobów z programów takich jak 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks i Allegorithmic Substance. Zasoby te mogą zostać dodane do projektu, po czym można nimi zarządzać z wnętrza Unity IDE poprzez graficzny interfejs użytkownika. Do definiowania shaderów wykorzystywany jest język ShaderLab, wspierający programowanie deklaratywne dla shaderów stałych, jak również shadery napisane w GLSL lub Cg. Shader może obsługiwać wiele wariantów a także zapewniać implementację fallback shadera, co pozwala Unity na wybranie najlepszego shadera dla danej karty graficznej a w przypadku braku wymaganej funkcjonalności wycofanie się do shadera zapasowego, który może poświęcać funkcjonalność w celu polepszenia wydajności aplikacji.

Programowanie odbywa się w jednym z trzech języków wspieranych przez silnik. Programiści mogą używać UnityScript (język podobny do JavaScript), C# lub Boo (język ze składnią zbliżoną do Pythona). Od wersji Unity 3.0 silnik rozpowszechniany jest ze zindywidualizowaną wersją MonoDevelop pomagającą w debugowaniu skryptów.

Unity wspiera wiele platform sprzętowych. Deweloperzy mają kontrolę nad wieloma aspektami dystrybucji swoich aplikacji na platformy takie jak przeglądarki, komputery osobiste, konsole, telefony komórkowe i tablety. Unity pozwala na wybranie szczegółów kompresji tekstur i rozdzielczości dla każdej ze wspieranych platform. Aktualnie wspierane są platformy takie jak BlackBerry 10, Windows 8, Windows Phone 8, Windows, Mac, Linux, Android, iOS, Unity Web Player, Adobe Flash, PlayStation 3, PlayStation Vita, Xbox 360, Wii U oraz Wii. W następnej wersji silnika zapowiedziano wsparcie dla urządzeń PlayStation 4 oraz Xbox One.

Unity posiada też wbudowane wsparcie dla silnika fizycznego PhysX firmy Nvidia z zaimplementowanym wsparciem dla symulacji tkanin w czasie rzeczywistym, raycastów z parametrem szerokości oraz obsługi warstw kolizyjnych. Unity 4.3 wprowadziło też obsługę obiektów fizycznych Box2D do wykorzystania w grach dwuwymiarowych.

## Web/HTML5/Flash

## 

# Wymagania przed-implementacyjne

Implementowane aplikacje testowe były tworzone w systemie Windows 7 i aktualizowane w systemie kontroli wersji git. Nie pozwalało to jednak na uruchamianie aplikacji na urządzeniach z systemami typu iPhone bądź Windows Phone 8, które do poprawnej kompilacji pliku binarnego stawiają dodatkowe wymagania pod względem bibliotek, które można znaleźć jedynie w systemach OS X lub Windows 8. Poniższe działy traktują o sposobie który został zastosowany w celu pominięcia tych wymagań.

## Windows Phone 8

W celu tworzenia aplikacji na telefony oparte o system Windows Phone 8 następujące kryteria muszą być spełnione:

* Windows 8 na maszynie, na której będzie tworzona aplikacja
* Unity 4.3+
* Windows Phone SDK 8.0+
* Zarejestrowany i odblokowany telefon z systemem Windows Phone 8+
* Konto Microsoft Account

### Instalacja Windows 8 w aplikacji Virtualbox

W celu spełnienia wymagania posiadania systemu Windows 8 w celu kompilowania konkretnych aplikacji do formy binarnej postanowiono zainstalować system w aplikacji Virtualbox.

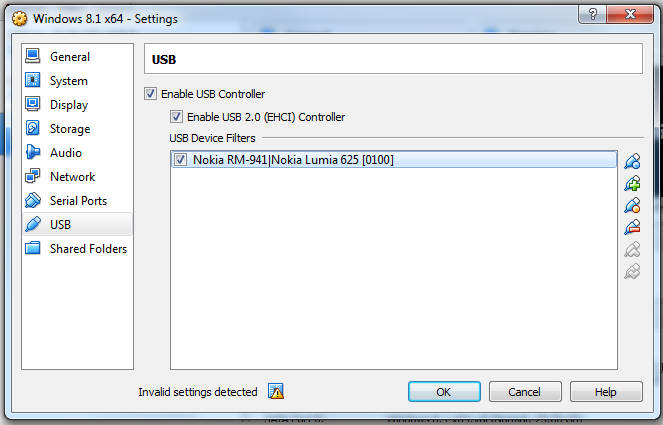
Jako obraz instalacyjny wykorzystano 90 dniową wersję ewaluacyjną systemu Windows 8.1 w formie 64 bitowej pobraną z oficjalnej strony firmy Microsoft. By Virtualbox dał możliwość instalacji wersji 64 bitowych systemów operacyjnych należy najpierw w ustawieniach BIOS danego urządzenia włączyć technologię wirtualizacji:

Zdjecie bios z virtualizacja

Po włączeniu wirtualizacji przystąpiono do tworzenia profilu systemu wirtualnego o następujących parametrach:

* Typ: Microsoft Windows
* Wersja: Windows 8.1 (64 bit)
* RAM: 2048MB
* Rozmiar dysku: 25GB
* Pamięć karty graficznej: 128MB

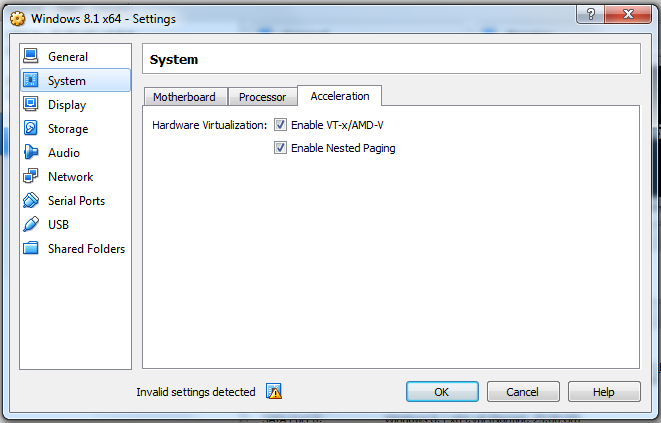
Po stworzeniu profilu systemu należało też wprowadzić kilka zmian konfiguracyjnych. Po pierwsze należało zainstalować kompatybilną wersję pakietu Virtualbox Extension Pack, która można znaleźć na oficjalnej stronie oprogramowania Virtualbox. Po instalacji pakietu rozszerzeń w ustawieniach wcześniej stworzonego profilu w zakładce USB należało zaznaczyć opcję „Enable USB 2.0 (EHCI) Controller” oraz dodać urządzenie „Nokia RM-941|Nokia Lumia 625 [0100]” do tabeli „USB Device Filters”.



Rysunek 3. Poprawnie skonfigurowana zakładka USB

Pierwsza ze zmian pozwala na wykrycie urządzenia przez system wirtualny. W systemie, który jest gościem urządzenia z Windows Phone 8 wykrywane są zawsze, jako urządzenia USB 2.0. Dodanie konkretnego urządzenia do filtrów USB programu Virtualbox informuje go, iż komunikacja USB powinna też być przesyłana do systemu wirtualnego, pozwalając na poprawne wykrycie i komunikację z telefonem.

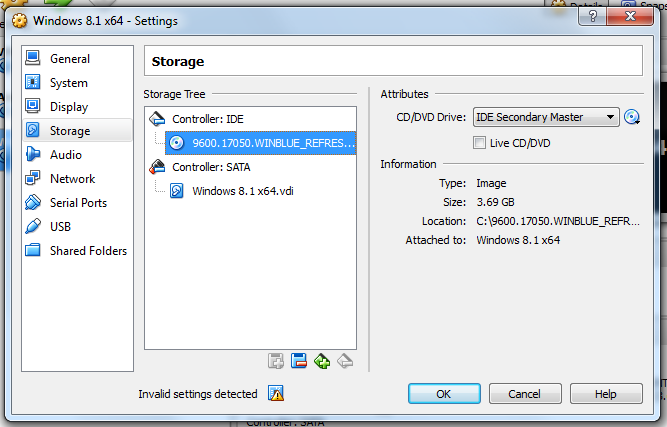
Kolejną zmianą konfiguracyjną wymaganą do poprawnego działania systemu wirtualnego było włączenie opcji „Enable VT-x/AMD-V” oraz „Enable Nested Paging” w zakładce „System”.



Rysunek 4. Konfiguracja zakładki system

Opcje te odpowiadają za umożliwienie aplikacji Virtualbox wykorzystania wirtualizacji sprzętowej systemu gospodarza, co znacznie poprawia szybkość działania systemu wirtualnego.

Ostatnią ze zmian konfiguracyjnych profilu było wskazanie obrazu, który system wirtualny powinien traktować jako znajdujący się w napędzie CD/DVD:



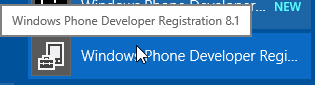
Rysunek 5. Wskazany obraz ISO systemu Windows 8.1

Po wskazaniu obrazu ISO z systemem Windows 8.1 przystąpiono do instalacji systemu. Po zakończeniu instalacji należało przeprowadzić dalszą konfigurację umożliwiającą kompilowanie aplikacji do formy binarnej a następnie przesyłanie jej do telefonu.

### Instalacja Windows Phone SDK 8 i rejestracja telefonu

Rozwinąć?

Po pobraniu i zainstalowaniu Windows Phone SDK 8.0 przystąpiono do procesu rejestrowania telefonu. W tym celu należy skorzystać z aplikacji Windows Phone Developer Registration 8.1 dostarczonej wraz z SDK:



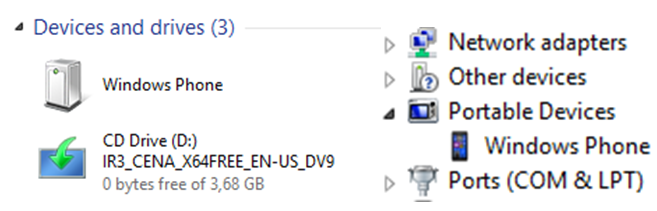
Rysunek 6. Aplikacja rejestrująca

Przed skorzystaniem z wyżej wymienionej aplikacji należy jednak przekierować komunikację telefonu do systemu goszczącego, poprzez wybranie odpowiedniej opcji w aplikacji Virtualbox:



Rysunek 7. Przekierowanie USB do systemu wirtualnego

Telefon powinien zostać wykryty przez system i sterowniki wymagane do jego rozpoznania powinny zostać zainstalowane automatycznie. Po zakończeniu tego procesu telefon powinien być widoczny w ustawieniach komputera:



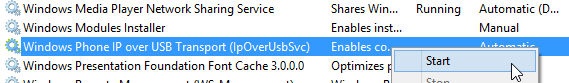
Rysunek 8. Urządzenie widoczne programach "Device Menager" i "This computer"

Gdy urządzenie jest już uruchomione należy przejść do wymienionego wcześniej Windows Phone Developer Registration 8.1 i przystąpić do odblokowania telefonu. Podczas uruchamiania programu może okazać się, że telefon nie zostaje wykryty:



Rysunek 9 Błąd w wykryciu urządzenia

Należy wtedy uruchomić program services.msc i aktywować usługę IpOverUsbSvc



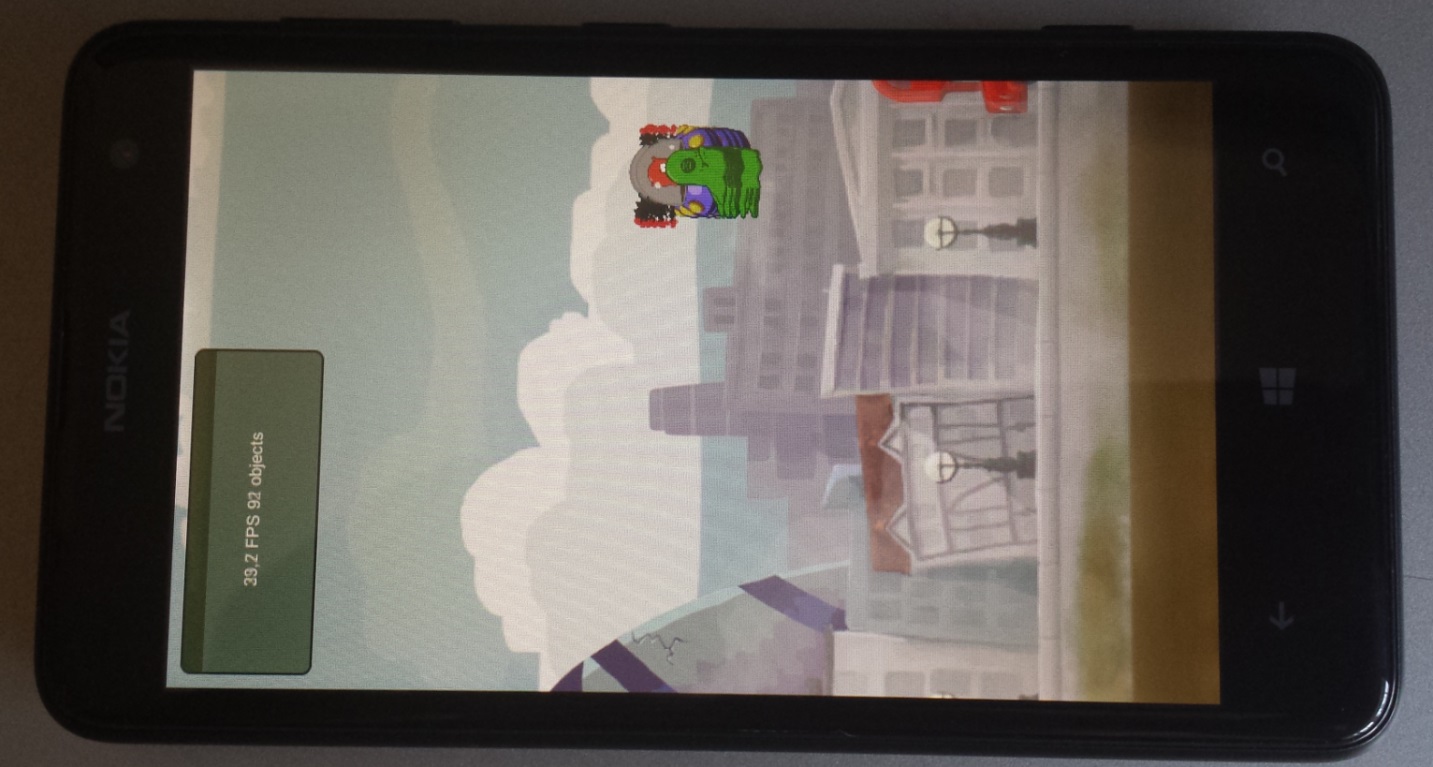
Rysunek 10 Aktywacja usługi IpOverUsbSvc

Po uruchomieniu usługi i zalogowaniu się na konto Microsoft, a następnie odblokowaniu ekranu urządzenia, telefon powinien zostać odblokowany dla aplikacji zewnętrznych:



Rysunek 11 Informacja o poprawnym odblokowaniu urządzenia

Następnie można już przejść do aplikacji Unity i po załadowaniu przykładowego projektu i sceny wybrać opcję „Build and Run for Windows Phone 8”:



Rysunek Aplikacja testowa uruchomiona na telefonie Nokia Lumia 625

# Zaimplementowane programy testujące

## Unity: Test grafiki 2D

W celu przetestowania wydajności urządzeń i sprawdzeniu możliwości silnika zaimplementowano prostą grę zręcznościową. Gra polega na wskakiwaniu po platformach na jak największą wysokość. Gdy gracz spadnie postać otwiera spadochron i zaczyna „zbierać” przeciwników. Po odzyskaniu kontaktu z podłożem zebrani przeciwnicy rozlatują się w kształt stożka. Zasoby wykorzystane w grze pochodzą z oficjalnej strony Unity.



Rysunek 13. Zrzut ekranu z gry zręcznościowej

### Implementacja

Każdy element w grze poczynając od tła, przez platformy, ikonki i postać gracza są typem GameObject. Klasa ta nie wykonuje żadnych czynności sama z siebie, można o niej myśleć, jako o kontenerze na inne obiekty, które nadają mu pewne zachowania i wygląd. W celu ułatwienia testowania gry stworzono kilka różnych skryptów obsługujących zachowania w różnych fazach gry. Skrypty te są przypisane do gracza tak długo jak jest w danej fazie. Gdy następuje przejście do innej fazy stary skrypt jest odpinany i następuje inicjalizacja nowego. Takie podejście pozwala na testowanie pojedynczych elementów gry bez rozpoczynania jej od początku za każdym razem i konieczności przejścia przez początkowe fazy w celu testowania funkcjonalności w fazach późniejszych. By jeszcze ułatwić testowanie stworzono też dodatkowe sceny odpowiedzialne za rozpoczęcie każdej z faz.

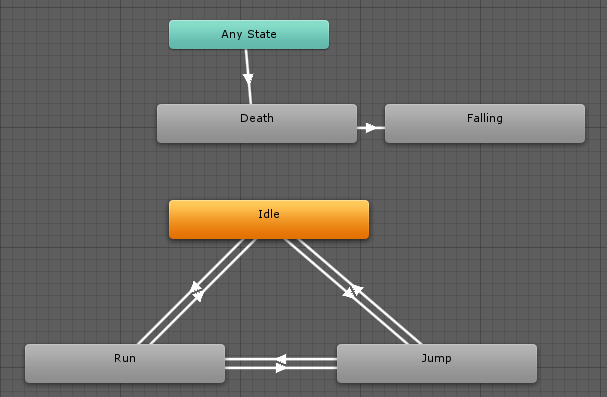
#### Postać gracza

Podstawowymi komponentami bohatera są obiekty klas RigidBody2D i Collider. Pierwsza z klas nadaje bohaterowi odpowiednie atrybuty fizyczne, takie jak masę czy współczynnik tarcia odpowiadający za stopień wytracania prędkości podczas chodu. Odpowiada też za dokładność obliczeń kolizji fizycznych, decydując czy obiekt liczy kolizje w sposób ciągły czy dyskretny. Drugi z komponentów decyduje o rozpatrywaniu kolizji z obiektami takimi jak platformy, przeciwnicy czy ograniczenia ekranowe. Każdy z obiektów klasy Collider informuje skrypty przypięte do bohatera o kolizjach z otoczeniem poprzez odpowiednie funkcje.



Rysunek 14. Postać gracza. Na zielono obiekty klasy Collider

Postać gracza zawiera też obiekt klasy AudioSource by mógł wydawać dźwięki, gdy np. skacze, oraz w obiekt Animator, który odpowiada za wywoływanie funkcji zmieniającej wygląd postaci podczas czynności takich jak bieganie czy skakanie.



Rysunek 15. Graf animacji gracza z odpowiednimi przejściami.

## Unity: Test ilości obiektów 2D

Kolejnym z zaimplementowanych testów był test możliwości tworzenia i destrukcji obiektów klasy GameObject. W grach zręcznościowych często występuje konieczność szybkiego tworzenia i niszczenia obiektów, które reprezentują przeciwników, pociski, elementy tła lub punktację. Wiedza na temat tego, ile takich operacji możemy przeprowadzić w danej klatce bez spowalniania wykonania aplikacji na różnych platformach jest bardzo cenna.

Opisać użyteczność w normalnej grze

Kod testujący polegał na pomiarze ilości klatek na sekundę (FPS - frames per second) w grze podczas tworzenia i niszczenia obiektów. Za granicę potrzebną do płynnego działania programu obrano 30fps [przypis dlaczego?]. Program będzie próbował dojść do ilości obiektów na ekranie, która pozwoli utrzymać wymaganą prędkość działania a następnie pokaże ilość stworzonych obiektów. Każdy z obiektów tworzonych i niszczonych podczas tego testu posiada w sobie obiekty RigidBody i Collider, choć kolizje są wyłączone.

Android: 30fps to ok 70 (development) 80 (release) standardowych obiektów na Galaxy S4

WindowsPhone (Lumia 625) – 90 obiektów w debug,

Optymalizacje? Pooling, albo zamiast tworzyć obiekty uzyc particle emitera?

Po przeprowadzeniu testu ze standardowym niszczeniem obiektów spróbowano wytestować też obiekty klasy ParticleEmitter pozwalające na tworzenie większej ilości obiektów ale ograniczających ich zachowanie np. do opadania i kolizji. Tworzenie obiektów w ten sposób i dołączanie do nich elementów typu RigidBody byłoby bardzo trudne i nieefektywne.

# Wyniki

# Podsumowanie i wnioski

# Literatura