AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PRACA DYPLOMOWA MAGISTERSKA

pt.

„Analiza możliwości implementacji oprogramowania

do wizualizacji 2D i 3D dla różnych platform mobilnych”

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: prof. dr hab. inż. Mirosław Głowacki

Podpis dyplomanta: Podpis promotora:

Kraków 2014

***(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 4](#_Toc395003108)

[2 Analiza problemu 5](#_Toc395003109)

[3 Systemy 8](#_Toc395003110)

[3.1 iOS 8](#_Toc395003111)

[3.1.1 Tworzenie i dystrybucja aplikacji na system iOS 10](#_Toc395003112)

[3.1.2 Instalacja aplikacji z systemu Windows 12](#_Toc395003113)

[3.2 Android 15](#_Toc395003114)

[3.2.1 Aktualizacje 17](#_Toc395003115)

[3.2.2 Architektura systemu 17](#_Toc395003116)

[3.2.3 Tworzenie aplikacji Android 20](#_Toc395003117)

[3.3 Windows Phone 22](#_Toc395003118)

[3.3.1 Tworzenie aplikacji na Windows Phone 8 23](#_Toc395003119)

[4 Środowiska międzyplatformowe 27](#_Toc395003120)

[4.1 Unity 27](#_Toc395003121)

[4.2 Marmelade SDK 29](#_Toc395003122)

[4.3 Engine 31](#_Toc395003123)

[4.4 33](#_Toc395003124)

[5 Zaimplementowane programy testujące 34](#_Toc395003125)

[5.1 Unity: Test grafiki 2D i 3D - benchmarki 34](#_Toc395003126)

[5.2 Unity: Gra 2D 42](#_Toc395003127)

[5.3 Unity: Curveball 44](#_Toc395003128)

[6 Podsumowanie i wnioski 50](#_Toc395003129)

[7 Literatura 52](#_Toc395003130)

# Wstęp

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 dostają do dyspozycji wiele nowych funkcjonalności, w szczególności wspierających mobilność konsumenta. Dzięki tym aplikacjom 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.

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. W tabeli 1. przedstawiono kilka różnic pomiędzy najpopularniejszymi mobilnymi systemami operacyjnymi.

Tabela 1. Zestawienie różnych cech systemów mobilnych

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| OS | VM | Język | GUI | Pamięć | IDE | Platformy | Urządzenia |
| iOS | Brak | Objective-C | Cocoa touch | Ref. count. | XCode | Os X | Homogeniczne |
| Android | Dalvik | Java | Pliki XML | GC | Eclipse | Wiele | Heterogeniczne |
| Windows Phone 8 | CLR | C# + .NET | Pliki XAML | GC | Visual Studio | Windows | Homogeniczne |
| BlackBerry | Java ME | Java | W kodzie | GC | Eclipse | Wiele | Heterogeniczne |
| Symbian | Dostępna | C++ | Qt | Ręczne | Qt Creator | Wiele | heterogeniczne |

Celem niniejszej pracy jest znalezienie środowiska umożliwiającego uruchomienie stworzonych aplikacji na kilku najpopularniejszych platformach mobilnych i przetestowanie ich wydajności. Stworzone zostaną benchmarki umożliwiające sprawdzenie optymalnych parametrów dla poszczególnych urządzeń, oraz przykładowe aplikacje, posiadające możliwość uruchomienia na jak największej ilości platform.

W pierwszej kolejności przedstawione zostaną platformy iOS, Android oraz Windows Phone 8 wraz ze sposobem konfiguracji ich środowisk deweloperskich. Następnie przybliżone zostanie kilka środowisk pozwalających na tworzenie aplikacji międzyplatformowych uruchamianych natywnie na danej architekturze. Po wybraniu jednego ze środowisk zaimplementowane zostaną aplikacje testujące wydajność poszczególnych urządzeń oraz przykładowe aplikacje prezentujące możliwości danego środowiska. Następnie przedstawione zostanie podsumowanie oraz wnioski płynące z niniejszej pracy.

# Systemy

Omawiane systemy wybrano w oparciu o analizę rynku urządzeń mobilnych na podstawie największego udziału systemów operacyjnych w sprzedanych nowych urządzeniach w danym przedziale czasowym[[1]](#footnote-1), co przedstawiono na wykresie 1.

Wykres 1

Obrano trzy najbardziej popularne systemy operacyjne: Android, iOS i Windows Phone.

## 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 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.

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.

Mechanizmem odpowiedzialnym za renderowanie grafiki 3D w systemach iOS jest:

* OpenGL ES 1.1 dla urządzeń iPhone, iPad i iPodTouch
* OpenGL ES 2.0 dla urządzeń z systemem iOS 5 i późniejszych np. iPhone3GS, iPad i iPad Mini
* OpenGL ES 3.0 dla urządzeń z systemem iOS 7 i późniejszych np. iPhone 5S, iPad Air i iPad Mini z ekranem Retina

System iOS nakłada kilka restrykcji na aplikacje korzystające z renderowania poprzez OpenGL ES. Jedną z nich jest zakazanie renderowania w aplikacjach działających w tle. W przypadku próby takowej operacji wykonanie aplikacji zostanie wstrzymane i system zgłosi sytuację wyjątkową. Kolejnym z ograniczeń jest obsługa obliczeń równoległych w OpenGL ES. Aplikacje nie mogą korzystać z jednego kontekstu z dwóch różnych wątków w tym samym czasie. Należy zadbać też, by wątki czyściły kontekst po wykonaniu swojej pracy, jako że brak czyszczenia może powodować nieokreślony stan wykonania dla kolejnych operacji na danym kontekście.

### 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.

W celu tworzenia aplikacji na urządzenia oparte o system iOS należy spełnić następujące kryteria:

* karta kredytowa MasterCard lub Visa
* konto deweloperskie firmy Apple
* posiadać urządzenie firmy Apple z najnowszym systemem OS X
* posiadać urządzenie firmy Apple z systemem iOS, zarejestrowane jako urządzenie do użytku deweloperskiego
* zainstalowana aplikacja XCode z najnowszym iOS SDK

Do tworzenia aplikacji na system iOS wymagana jest aplikacja XCode, jej biblioteki oraz iOS SDK, które publikowane są tylko i wyłącznie dla systemów OS X i nie ma możliwości uruchomienia ich np. na systemach Windows czy Linux. Alternatywą dla kupienia komputera firmy Apple mogą wtedy być maszyny wirtualne takie jak np. Virtualbox. Jednak firma Apple zastrzegła w regulaminie, iż system OS X może być instalowany tylko i wyłącznie na urządzeniach tejże firmy. Istnieją rozwiązania pozwalające na instalację systemu OS X na zwykłych komputerach domowych, czy nawet jako urządzenia wirtualne w aplikacji Virtualbox, jednak autor by pozostać w zgodzie z EULA Apple postanowił skorzystać z usług firm wypożyczających maszyny wirtualne OS X, uruchomione na urządzeniach OS X. Usługi te udostępniają wirtualny pulpit wraz z dostępem administracyjnym, pozwalają na różną rozdzielczość płatności (co godzinę, tygodniowo lub miesięcznie) i są stosunkowo tanie – wahając się pomiędzy 20 a 60 dolarów za miesiąc użytkowania.

Dostarczone rozwiązanie zapewniało wersję 10.9 Mavericks systemu OS X wraz z dostępem administratora oraz 30Gb przestrzeni dyskowej.

Mając zapewniony dostęp do systemu OS X można przejść do instalacji potrzebnych narzędzi. Zaczynając instalację XCode, który niezbędny jest do poprawnego działania Unity dla iOS, należy przejść do aplikacji App Store i przeprowadzić wyszukiwanie po jego nazwie. Po wybraniu przycisku FREE jeżeli to nasza pierwsza wizyta w aplikacji App Store zostaniemy poproszeni o stworzenie konta App Id – uniwersalnego identyfikatora firmy Apple wykorzystywanego w aplikacjach takich jak na przykład iTunes. W tym momencie potrzebna będzie aktywna karta kredytowa zarejestrowana w kraju pochodzenia programisty. Firma Apple nie pozwala na tworzenie nowych kont bez podania danych karty płatniczej, adresu oraz innych danych użytkownika. Po poprawnym wypełnieniu formularza, zatwierdzeniu danych karty płatniczej oraz weryfikacji adresu e-mail stworzone zostanie konto App Id. Dopiero w tym momencie można wybrać w ustawieniach aplikacji App Store sposób płatności na None.

Jeżeli mamy już konto App Id i wybierzemy przycisk instalacji XCode powinniśmy zobaczyć postęp pobierania a następnie instalacji programu. Podgląd istnieje także w aplikacji Launchpad, w której należy wznowić operację pobierania gdyby została przerwana np. poprzez zrestartowanie systemu.

Gdy instalacja dobiegnie końca możemy przystąpić do uruchomienia aplikacji XCode w celu jej inicjalizacji. XCode pobierany z AppStore posiada wbudowany iOS SDK, więc możemy przystąpić do pobrania silnika Unity z oficjalnej strony aplikacji.

Unity dla systemów OS X pobierana jest, jako plik o rozszerzeniu \*.dmg, który to stanowi samodzielny instalator. Po zakończeniu instalacji w folderach Applications tworzony jest katalog Unity, z którego możemy uruchamiać aplikację. Po tych operacjach w systemie powinno być zainstalowane oprogramowanie potrzebne do tworzenia aplikacji na system iOS.

Firma Apple wymaga od deweloperów posiadania konta programisty – kosztującego 99$ rocznie – w celu dystrybucji aplikacji na prawdziwe urządzenia. Co ciekawe, konto to jest też wymagane do przełączenia telefonu w tryb deweloperski. Oznacza to, że bez ponoszenia kosztów finansowych nie można nawet przetestować swojej aplikacji na prawdziwym urządzeniu i korzystać można tylko z ograniczonego iOS Symulatora.

Autor niniejszej pracy pokusił się o znalezienie sposobu niewymagającego posiadania płatnego konta. W tym celu jednym z przedsięwziętych kroków było odblokowanie telefonu dla dostępu administracyjnego, czyli tzw. jailbreak. Opis tej operacji wykracza poza temat pracy, ale nie jest ona ciężka do wykonania. Należy jednak pamiętać, iż proces ten powoduje utratę gwarancji producenta, choć istnieją sposoby przywrócenia systemu do stanu pierwotnego.

Po uwolnieniu telefonu i uzyskaniu dostępu do funkcji administratora (root, często zwanym też super user) w pierwszej kolejności należało zainstalować aplikacje umożliwiające komunikację protokołem SFTP pomiędzy urządzeniem a komputerem osobistym. W tym celu na telefonie należało zainstalować paczki OpenSSL oraz OpenSSH.

### Instalacja aplikacji z systemu Windows

Mimo faktu, że systemy Windows nie są wspierane, jako platformy deweloperskie dla systemu iOS da się przeprowadzić kompilację oraz instalację własnej aplikacji na urządzeniu rzeczywistym poprzez zastosowanie odpowiednich programów.

Jednym z pakietów, który może w tym pomóc jest „iOS Build Enviroment for Windows developers”. Pakiet ten pozwala na kompilację aplikacji przeznaczonych na iOS pod systemem Windows, podpisanie aplikacji oraz uruchomienie jej na urządzeniach z systemem iOS. Pakiet pozwala też na uruchomienie zdalnej konsoli debugującej przedstawiającej błędy z urządzenia na komputerze osobistym, kompilację bezpośrednio z IDE Visual Studio czy też generację paczek \*.deb oraz \*.ipa. Aplikacja posiada też przydatną dla potrzeb niniejszej pracy funkcjonalność kompilowania projektów XCode generowanych przez platformę Unity bezpośrednio na maszynie, na której były pisane, bez konieczności eksportowania projektu na maszynę z systemem OS X. Do poprawnej inicjalizacji aplikacja potrzebuje jednak dostępu do plików iOS SDK, dlatego dostęp do systemu OS X z zainstalowanym XCode, będzie potrzebny przynajmniej na krótki czas.

Po pobraniu i standardowym procesie instalacji należy przeprowadzić konfigurację środowiska poprzez eksportowanie kilku części iOS SDK. W tym celu do pakietu dołączono dwa skrypty znajdujące się w folderze instalacyjnym w katalogu „Migration Assistant”. Jeden ze skryptów z rozszerzeniem \*.command przeznaczony jest dla systemu OS X. Wyszukuje on w systemie zainstalowany iOS SDK, następnie kopiuje z niego foldery „System” oraz „usr” pliki „libclang\_rt.ios.a”. Skopiowane pliki umieszczane są w pliku SDK.zip. Spakowane archiwum należy przenieść na maszynę z systemem Windows, na której odbywał będzie się proces kompilacji. Po przeniesieniu archiwum należy uruchomić drugi ze skryptów z rozszerzeniem \*.cmd, który przeniesie pliki skopiowane z SDK do odpowiednich katalogów i zmodyfikuje zmienne środowiskowe w sposób pozwalający z nich korzystać. Pomimo opisu samego programu mówiącego o korzystaniu ze skryptów z przenośnego urządzenia USB, można bez żadnych komplikacji korzystać z nich z dowolnego miejsca w systemie, do którego posiadamy dostęp typu odczyt-zapis.

Po dokonaniu konfiguracji aplikacji należy wygenerować projekt XCode z platformy Unity. By tego dokonać należy przejść do menu File->Build Settings, przełączyć platformę na iOS oraz wybrać przycisk Build.

Po tym zabiegu w wybranym folderze wygenerowany zostanie projekt XCode odpowiadający eksportowanemu projektowi Unity. Projekt ten należy wskazać aplikacji „unity-builder.exe” dostarczonej wraz z paczką iOS Build Enviroment, widocznej na rysunku .



Rysunek 1.Okno programu unity-builder.exe

Eksportowany projekt XCode należy wskazać poprzez wybranie go po naciśnięciu przycisku „Browse”. Opcja „Certificate to use” pozwala na podpisanie generowanej aplikacji jednym z kluczy przechowywanych w narzędziu „Keychain tool” , lub pseudo podpisanie kluczem prywatnym generowanym na komputerze użytkownika. „Rebuild all source files” wymusza na aplikacji budowanie projektu od nowa, ignorując pliki już wcześniej podpisane. Wydłuża to czas kompilacji, ale pozwala uniknąć potencjalnych błędów. „Deploy to iOS device over USB immediately” jest funkcjonalnością eksperymentalną pozwalającą na uruchomienie skompilowanej aplikacji bezpośrednio na urządzeniu połączonym z komputerem osobistym poprzez interfejs USB. „Also build .deb packages for jailbroken devies” dodaje krok do procesu budowania, który generuje paczki \*.deb (domyślnymi paczkami są \*.ipa), które można zainstalować na odblokowanym urządzeniu korzystając z programów Cydia lub dpkg. „Open packages folder after successfull build” otwiera folder w którym przechowywane są paczki \*.ipa i \*.deb po zakończeniu kompilacji. Dla potrzeb niniejszej pracy wykorzystano opcję pseudo podpisu, oraz generacji paczek \*.deb. Instalacji aplikacji na urządzeniu bezpośrednio po kompilacji nie działała, prawdopodobnie z powodu bardzo starej wersji wykorzystywanego do testów urządzenia.

Po zakończeniu kompilacji pojawi się okno przypominające o ograniczeniach programu oraz wygenerowane paczki:



Rysunek 2. Wygenerowane paczki \*.deb oraz \*.ipa

Posiadając gotowe do instalacji paczki należy połączyć się z urządzeniem poprzez SSH w celu przekopiowania ich do pamięci telefonu. Dla celów pracy dokonano tego poprzez program WinSCP oraz opisane wcześniej paczki OpenSSL i OpenSSH zainstalowane na telefonie. By poprawnie połączyć się z urządzeniem typu iPhone należy skorzystać z protokołu SFTP. Domyślnie wszystkie odblokowane urządzenia z OpenSSH przyjmują autoryzację przy pomocy użytkownika „root” z hasłem „alpine”.

Mając otwartą sesję do urządzenia należy przekopiować paczki. Instalacji można dokonać z konsoli danego telefonu poprzez komendę „dpkg”. Dla potrzeb pracy jednak instalacja przebiegła przez aplikację Cydia, znacznie ułatwiającą wszelkie interakcje z paczkami. W celu instalacji paczki \*.deb poprzez aplikację Cydia należy skopiować paczkę do katalogu /private/var/root/Media/Cydia/AutoInstall. Jeżeli katalog Cydia nie istnieje w katalogu Media należy go stworzyć i tak samo postąpić z katalogiem AutoInstall.

Po umieszczeniu paczki w odpowiednim miejscu należy zrestartować urządzenie dwa razy. Po drugim restarcie nasza aplikacja pojawi się w menu Spring urządzenia.

## 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).

Mechanizmem odpowiedzialnym za renderowanie w systemach Android jest:

* OpenGL ES 1.0 i 1.1 na urządzeniach z systemem Android 1.0 wzwyż
* OpenGL ES 2.0 na urządzeniach z systemem Android 2.2 (API 8) wzwywż
* OpenGL ES 3.0 na urządzeniach z systemem Android 4.3 (API 18) wzwyż

Należy pamiętać, iż odpowiednia implementacja OpenGL ES musi być dostarczona przez producenta danego urządzenia by móc skorzystać z jej funkcjonalności. Znaczy to, że nie każde urządzenie z systemem 4.3 obsługiwało będzie OpenGL ES 3.0.

Jak widać na załączonym rysunku 1. OpenGL ES dostarczany jest jako biblioteka i aplikacje nie mają bezpośredniego kontaktu z samą implementacją. Android pozwala na komunikację z implementacją OpenGL ES przez wyspecjalizowane API jak i przez NDK (Native Development Kit), które pozwala na komunikację z rendererem w języku C\C++. Każda z aplikacji chcąc korzystać z funkcjonalności OpenGL ES musi zadeklarować taką chęć w pliku manifestu aplikacji. Aplikacje określone jako wymagające OpenGL ES będą filtrowane w sklepie Google Play by uniknąć sytuacji pobrania aplikacji na urządzenie nie wspierające danej funkcjonalności. Jako, że OpenGL ES 3.0 jest kompatybilny wstecznie z wersją 2.0 możliwe i zalecane jest określanie aplikacji jako wymagającej do działania OpenGL ES w wersji 2.0, a następnie, już w czasie uruchomienia aplikacji, sprawdzenie obsługi wersji 3.0 przez urządzenie i ewentualne korzystanie z funkcjonalności w razie ich wspierania.



Rysunek 3. Architektura systemu Android

### Tworzenie aplikacji Android

W celu tworzenia aplikacji na telefony oparte o system Android następujące kryteria muszą być spełnione:

* Zainstalowane Android SDK
* Telefon z włączonymi opcjami programisty

W celu tworzenia aplikacji na urządzenia z systemem Android wymagane jest zainstalowanie Android SDK. Najwygodniejszym rozwiązaniem jest pobranie tegoż w paczce ADT Eclipse Bundle wraz z skonfigurowanym IDE Eclipse.

Po pobraniu paczki należy ją wypakować i przystąpić do konfiguracji oraz pobrania odpowiednich platform, narzędzi oraz wersji Androida. W tym celu należy uruchomić program „SDK Manager”dostarczony w paczce. Po uruchomieniu należy wybrać interesujące nas platformy i narzędzia, w tym przypadku „Android 4.2.2 (API17)”, oraz zainstalować „Google USB Driver”z paczki „Extra”. Po przeczytaniu i zatwierdzeniu licencji program powinien przystąpić do pobierania wskazanych paczek.

Po ukończeniu działania narzędzia należy przejść do środowiska Unity i wskazać katalog z Android SDK. Wyboru dokonujemy w edytorze Unity w manu Edit->Preferences… w zakładce External Tools.

By móc testować aplikacje na rzeczywistym urządzeniu należy włączyć usługę „USB debugging” w opcjach programisty. W celu przejścia do opcji programisty w telefonach z systemem Android należy postępować zgodnie z poniższym schematem, w zależności od wersji systemu:

W urządzeniach z systemem Android 2.1 do 2.3.6 opcje deweloperskie widoczne były w menu Settings> Applications> Development.

W wersjach 4.0 oraz 4.1 menu zostało przeniesione i jego ścieżka znajdowała się w Settings> Developer Options.

Od systemu 4.2 wzwyż menu opcji programisty zostało ukryte by uniknąć przypadkowego nadpisania ważnych ustawień urządzenia przez początkujących użytkowników. By odblokować menu w wyżej wymienionych systemach należy przejść do menu Settings -> About device i przewinąć do informacji Build nuber. Następnie wymienioną opcję należy kliknąć siedem razy, co potwierdzone zostanie informacją „Developer mode has been enabled” lub „You are now a developer!”:



Rysunek 4. Poprawnie przeprowadzony proces odblokowywania opcji programisty na urządzeniu z systemem 4.2.2

Po wykonaniu tych czynności w menu Settings odblokowana zostanie opcja Developer options, w której dla potrzeb tej pracy najbardziej interesującą opcją jest USB debugging. Po odblokowaniu opcji deweloperskich i uruchomieniu usługi USB debugging telefon gotowy jest na integrację ze środowiskiem Unity i ładowanie aplikacji testowych z pełnym wsparciem dla wbudowanego w engine debuggera.

## Windows Phone

Windows Phone 8 jest systemem trzeciej generacji z rodziny systemów mobilnych Windows Phone firmy Microsoft. Został upubliczniony 29 października 2012 roku i jak poprzednik posiadał interfejs Modern UI (wcześniej Metro). Kolejną wersję systemu – Windows Phone 8.1 – wydano 2 kwietnia 2014 roku. System zastępuje architekturę Windows-CE wykorzystaną w Windows Phone 7 nowym kernelem Windows NT stosowanym również w systemach desktopowych Windows 8. Obecnie urządzenia z Windows Phone 7 nie mają możliwości aktualizacji do nowego systemu ani uruchamiania aplikacji skompilowanych z myślą o tym systemie.

Windows Phone 8 posiada obsługę ekranów w wyższych rozdzielczościach, obsługę procesorów wielordzeniowych, NFC, wsteczną kompatybilność z aplikacjami Windows Phone 7, lepsze wsparcie dla pamięci zewnętrznych (które teraz obsługiwane są podobnie jak w Androidzie lub wersji systemu na komputery osobiste) oraz integrację sprzętową z aplikacjami VoIP. Posiada również więcej funkcjonalności zaprojektowanych z myślą o przedsiębiorstwach takie jak zewnętrzne zarządzanie urządzeniami, szyfrowanie danych oraz możliwość stworzenia wewnętrznego marketu aplikacji, w celu dystrybuowania aplikacji wewnętrznych dla pracowników. Dodatkowo Windows Phone 8 wspiera aktualizację poprzez interfejsy bezprzewodowe. Wszystkie urządzenia z systemem wspierane będą przez minimum 36 miesięcy po wejściu do sprzedaży.

Windows Phone 8 jest pierwszym mobilnym systemem firmy Microsoft korzystającym z kernela Windows NT. Funkcjonalności systemu względem swoich poprzedników zostały rozwinięte o poprawioną obsługę plików, stosu sieciowego, komponentów zabezpieczeń czy też obsługi grafiki. Dzięki adaptacji nowego kernela system posiada wsparcie dla procesorów wielordzeniowych, wyższe od poprzedników rozdzielczości ekranu oraz obsługę kart MicroSD.

W przeciwieństwie do poprzedników Windows Phone 8 posiada prawdziwe możliwości wielozadaniowości, gdzie programiści mogą tworzyć aplikacje uruchamiane w tle potrafiące przywracać się na pierwszy plan bez konieczności ponownej inicjalizacji. Użytkownik może przełączać się pomiędzy aktywnymi aplikacjami poprzez przytrzymanie przycisku Wstecz, gdzie ma też możliwość uśpienia lub zakończenia dowolnej z aplikacji. W pewnych warunkach, takich jak błędy w komunikacji sieciowej czy też niski stan baterii, system sam może zdecydować o zawieszeniu działania pewnych aplikacji działających w tle, szczególnie jeżeli użytkownik nie korzystał z nich od dłuższego czasu. Windows Phone 8 pozwala też na kontrolowanie urządzenia typu Xbox 360 lub Xbox One. W wersji 8.1 systemu dodano możliwość korzystania z VPN.

### Tworzenie aplikacji na 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

W celu spełnienia wymagania posiadania systemu Windows 8 dla możliwości 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.

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”. 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”. 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. Dokonać można tego w zakładce „Storage” wybierając jeden z kontrolerów IDE.

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.

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.

Przed skorzystaniem z wyżej wymienionej aplikacji należy jednak przekierować komunikację telefonu do systemu goszczącego, poprzez wybranie odpowiedniej opcji w aplikacji Virtualbox jak pokazano na rysunku 14.



Rysunek 5. Przekierowanie USB do systemu wirtualneg

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.

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 6. Błąd w wykryciu urządzenia

Należy wtedy uruchomić program services.msc i aktywować usługę IpOverUsbSvc. Po uruchomieniu usługi i zalogowaniu się na konto Microsoft, a następnie odblokowaniu ekranu urządzenia, telefon powinien zostać odblokowany dla aplikacji zewnętrznych.

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

# Środowiska międzyplatformowe

W poniższym dziale wymieniono kilka najpopularniejszych środowisk międzyplatformowych w celu wybrania najbardziej odpowiedniego do niniejszej pracy. Pod uwagę brano takie czynniki jak natywny język programowania w danym środowisku, koszty rozpoczęcia tworzenia oprogramowania, wsparcie społeczności deweloperskich każdej z aplikacji czy też stopień ich udokumentowania.

## 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 7. 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 Phone, 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.

## Marmelade SDK

Marmalade SDK (wcześniej Airplay SDK) jest międzyplatformowym silnikiem służącym do tworzenia oprogramowania i gier firmy Marmelade Technologies Limited, posiadającym pliki biblioteczne, przykłady, dokumentację i narzędzia wymagane do tworzenia, testowania oraz publikacji aplikacji dla urządzeń mobilnych.



Rysunek 8. http://uploads.gamedev5.net/gallery/album\_371/gallery\_25962\_371\_41468.png

Jak przedstawiono na rysunku 8. Marmelade nie posiada zintegrowanego edytora, ale potrafi korzystać z możliwości Visual Studio. Marmelade posiada symulatory dla większości platform które wspiera, pozwalające na testowanie aplikacji bez konieczności interakcji z rzeczywistym urządzeniem.

Podstawowym założeniem Marmelaed SDK jest możliwość jednokrotnego napisania tworzonej aplikacji i uruchomienie jej na wielu platformach. Efekt ten osiągnięto poprzez użycie C/C++ jako warstwy abstrakcyjnej dla API wystawionego poprzez każdą z wspieranych platform. W zależności od zakupionej licencji Marmelade SDK pozwala na tworzenie aplikacji na następujące platformy: Android, BlackBerry, Ios, LG Smart TV, Tizen, Mac OS X, Windows Desktop, Roku 2 i 3 oraz Windows Phone 8.

Marmelade SDK składa się z dwóch podstawowych warstw. Warstwa Marmelade System to niskopoziomowe API C pozwalające na dostęp do funkcjonalności urządzenia takich jak zarządzanie pamięcią, dostęp do plików, funkcji sieciowych, akcelerometrów, ekranu dotykowego, dźwięku oraz ekranu. Marmelade Studio to API C++ skupiające się na funkcjonalności wyższego rzędu głównie skupionej wokół obsługi renderowania 2D i 3D, zarządzania zasobami danej aplikacji oraz obsługą funkcji sieciowych. Bardzo znaczącą funkcjonalnością tego pakietu jest możliwość kompilowania tworzonej aplikacji na urządzenia z systemem iOS i Mac OS bez konieczności posiadania urządzeń firmy Apple, acz publikacja do oficjalnego sklepu internetowego dalej wymaga podpisania aplikacji na sprzęcie Apple.

Marmelade SDK pozwala na obsługę renderowania przez urządzenie mobilne zarówno poprzez OpenGL ES API, jaki i przez warstwę Marmelade Studio. Warstwa ta pozwala na korzystanie z ładowania i renderowania zasobów graficznych takich jak bitmapy i modele trójwymiarowe. Pozwala też na korzystanie z wtyczek do programów takich jak Autodesk 3DS Max i Autodesk Maya co pozwala na bezproblemowe korzystanie z obiektów trójwymiarowych, wraz z ich animacjami, w tworzonych aplikacjach. W celu wsparcia starszych urządzeń, nieposiadających osobnych modułów odpowiedzialnych za renderowanie grafiki, Marmelade SDK wprowadza moduł wsparcia wstecznego.

## jMonkey Engine

jMonkeyEngine (jME) jest silnikiem służącym do tworzenia trójwymiarowych i dwuwymiarowych aplikacji i gier działających na wielu platformach. jMonkeyEngine napisany jest w Javie i domyślnie korzysta z LWJGL jako renderera, choć istnieje możliwość korzystania z renderera opartego na JOGL. Silnik w pełni wspiera OpenGL 2,3 i 4, oraz OpenGL ES 2.0 i 3.0. Środowisko najczęściej dostarczane jest, jako zintegrowana wtyczka do IDE NetBeans, co widać na rysunku 9.



Rysunek 9. http://twolofbees.com/uploads/temp/dance/jmonkey\_test1.jpg

jMonkeyEngine jest projektem open source publikowanym pod nową licencją BSD. Korzysta z niego kilka komercyjnych firm deweloperskich i uczelni. Sam silnik jest zbiorem bibliotek i narzędzi, składających się na niskopoziomową aplikację służącą do tworzenia gier. W Połączeniu z IDE, jak np. ma to miejsce w oficjalnym jMonkey Engine 3 SDK, staje się wysokopoziomowym narzędziem z wieloma komponentami graficznymi. Integracja ze środowiskiem NetBeans pozwala na korzystanie z możliwości edytorów graficznych a także pozwala na skorzystanie z modularności samego IDE poprzez tworzenie nowych wtyczek. Obok domyślnych aktualizacji samego środowiska NetBeans, jMonkeyEngine wprowadza swoje własne repozytoria z wtyczkami, w wersjach stabilnych a także kompilowanych co noc.

Sam silnik daje do dyspozycji deweloperów wiele narzędzi, poczynając od klas odpowiedzialnych za obsługę operacji wejścia/wyjścia, widoków na ekranie, czy renderowania prostych kształtów, na obliczeniach fizycznych i komunikacji sieciowej kończąc. jMonkey Engine rozpoznaje wiele zasobów wykorzystywanych w produkcji gier komputerowych, takich jak modele stworzone w aplikacji Blender, czy też wszystkie zasoby zgodne ze specyfikacją OGRE. Obliczeniami fizycznymi zajmuje się silnik Bullet w wersji przystosowanej do użytkowania w programach napisanych w Javie. Do samej platformy dołączonych jest wiele klas zajmujących się obsługą często powtarzających się czynności w grach komputerowych. Przykładami takich klas mogą być CharacterController zajmujący się obsługą domyślnego poruszania się postaci wraz ze skokiem, przyspieszeniem i bezwładnością, czy też TerrainLodControl pozwalający na automatyczne zmienianie jakości generowanego terenu w zależności od odległości od kamery. Szeroka biblioteka klas bardzo pomaga w szybkim pisaniu i testowaniu tworzonych aplikacji.

## Podsumowanie

Z przedstawionych środowisk zdecydowano się wybrać Unity 3D. Argumentami przemawiającymi za Unity były głownie prostota w implementacji, duże wsparcie środowiska deweloperów (fora, dokumentacja, samouczki w formie wideo) oraz możliwość dostarczenia implementacji na platformy opisane w niniejszej pracy bez konieczności ponoszenia dodatkowych opłat już w momencie pisania aplikacji.

W porównaniu do Marmelade SDK Unity nie wymaga opłat za każdą z platform dystrybucji, ale nie posiada też bardzo pożądanej funkcjonalności kompilacji kodu iOS na platformach Windows. Marmelaed SDK nie posiada wsparcia dla starszych urządzeń typu iOS oraz, mimo że nie jest to tematem pracy, dla systemów z rodziny Linux. Znaczący był też fakt iż Marmelade SDK używa C++ do tworzenia wszelkich skryptów, gdzie C# wykorzystywany w Unity jest znacznie prostszym językiem zarówno do nauczenia jak i do późniejszej pracy.

W porównaniu do jMonkey Engine Unity wydaje się być bardziej stabilną, dojrzalszą platformą wspieraną przez firmę i regularnie aktualizowaną. jMonkey Engine nie posiada tak szerokich zasobów treningowych jak Unity i dokumentacja często nie nadąża za implementacją, o czym dowiadywać trzeba się z forów samego silnika. jMonkey Engine nie posiada też od razu działającego systemu przerzucania aplikacji na urządzenie rzeczywiste, wymagając instalacji dodatkowych sterowników. Brak jest też wsparcia dla systemu Windows Phone 8. Dużym plusem platformy jest brak konieczności płacenia za wykorzystanie silnika w przypadku publikacji aplikacji, acz jest to zaleta niezmieniająca oceny silnika w kontekście pracy.

# Zaimplementowane programy testujące

## Unity: Test grafiki 2D i 3D - benchmarki

Pierwszymi z zaimplementowanych programów była grupa benchmarków stworzona w celu przetestowania możliwości sprzętowych posiadanych urządzeń. Posłużono się w tym celu sposobem testowania zaproponowanym na stronie Juicbox games <http://blog.juiceboxmobile.com/2013/03/06/2d-gaming-mobile-performance-starling-air-vs-unity-3d/> . Na rysunku 10. przedstawiono diagram klas stworzonych testów.



Rysunek 10. Diagram klas aplikacji benchmarkowej

Stworzone sceny mają w sobie obiekt kamery, obiekt odpowiedzialny za tworzenie odpowiedniego testu oraz przycisk przełączający na dla wyświetlenie kolejnego testu. Obiekt testujący składał się ze skryptu opisującego dany test oraz obiektu typu TestEngine, w którym przechowywane były metody odpowiedzialne za tworzenie konkretnych obiektów już podczas trwania benchmarków. Skrypt testujący klasy TestMain posiadał między innymi referencję na obiekt TestEngine, kolejkę testów czekających na wykonanie oraz słownik, w którym przechowywane były wyniki testów. Opisana też była w nim logika odpowiedzialna za przełączanie kolejnych testów oraz metoda HandleEvent() zajmująca się zapisywaniem wyników każdego z benchmarków oraz metoda handleEnd() odpowiedzialna za wypisanie tychże po wyczerpaniu testów z kolejki.

Klasa TestEngine posiadała między innymi referencję na tworzone obiekty (czy to dwu czy trójwymiarowe), listę stworzonych obiektów wykorzystywaną przy ich późniejszym niszczeniu oraz funkcje odpowiedzialne za inicjalizację, pomiar czasu trwania testu oraz późniejsze posprzątanie środowiska przed rozpoczęciem kolejnego testu. Klasa ta posiada też moduł pozwalający na sztuczne obciążenie urządzenia operacjami zmiennoprzecinkowymi poprzez ustawienie zmiennej „workload”.

 Podczas sprzątania klasa kontaktuje się też z klasą TestMain poprzez interfejs IEvent przesyłając obiekt typu BenchmarkEvent w celu poinformowania tejże o zakończeniu danego testu, oraz przekazania wyników.

Klasa TestEngine odpowiedzialna jest za tworzenie obiektów typu TileClass dla testów dwuwymiarowych oraz BoxClass dla testów trójwymiarowych. Obiekty klasy TileClass posiadają w sobie metody odpowiedzialne za przemieszczanie każdego z paneli oraz za losową zmianę wyświetlanej na panelu tekstury. Klasa posiada zaimplementowane zachowania pozwalające jej na podjęcie decyzji czy dany panel powinien być animowany, czy i z jaką prędkością powinien się poruszać oraz w wypadku obiektów trójwymiarowych – czy powinien obracać się wokół własnej osi.

By dodać testy do kolejki wykonania należy w klasie TestMain wywołać metodę addTest() z parametrami reprezentującymi nazwę testu, ilością tworzonych obiektów typu TileClass lub BoxClass oraz z parametrami decydującymi czy test ma dodatkowo obciążać urządzenie operacjami zmiennoprzecinkowymi i jeżeli tak – jaką ich ilością.

Na urządzeniach zaprojektowano i uruchomiono testy dla 50, 250 i 500 obiektów nieruchomych, animowanych w celu sprawdzenia wydajności procesorów graficznych dostępnych w urządzeniach. Następnie dla takich samych ilości obiektów stworzono test z poruszającymi się obiektami. Zaimplementowano też testy dla 100 obiektów ruchomych, animowanych z dodanym obciążeniem operacjami zmiennoprzecinkowymi w ilościach pomiędzy 100 a 2500 operacji na klatkę. Należy pamiętać, że operacje te wykonywane są przez instancję klasy „TestEngine” więc nie są zależne od ilości obiektów na ekranie. Stworzone testy zostały też zduplikowane w celu stworzenia serii testów dla obiektów trójwymiarowych. Parametry są identyczne, zmieniany jest tylko tworzony obiekt. Stworzono także test inkrementalny zwiększający ilość obiektów przedziałami od 250 do 2000 obiektów w celu znalezienia optymalnej ilości obiektów na ekranie na poszczególnych urządzeniach. Ostatnim 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. Kod testujący polegał na pomiarze ilości klatek na sekundę w grze podczas tworzenia i niszczenia obiektów. Za granicę potrzebną do płynnego działania programu obrano 30fps [przypis dlaczego?]. Program inkrementacyjnie zwiększa ilość tworzonych oraz niszczonych 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.

Wszystkie benchmarki oraz stworzone aplikacje testowane były na urządzeniach:

* Samsung Galaxy S4 GT-I9505 – Android 4.2.2
* Nokia Lumia 625 – Windows Phone 8
* Samsung Galaxy Core Plus SM-G350 – Android 4.2.2
* iPhone 5 – iOS 7.1.2
* iPad 3 – iOS 7.1.2
* iPad 1 – iOS 5.1.1

Wykres 2

Jak widać na wykresie 2. żadne z urządzeń nie miało problemów z utrzymaniem zadowalającej liczby klatek na sekundę przy 50 obiektach widocznych na ekranie. Znaczący spadek wydajności widać za to już przy 250 obiektach, gdzie urządzenie z systemem Windows Phone 8 z ledwością przekroczyło próg 30 klatek. Przy 500 obiektach na ekranie jedynie urządzenia iPhone 5 i iPad 3 posiadały wystarczającą wydajność by zapewnić płynność testów. Co ciekawe, telefon Samsung Galaxy Core Plus utrzymywał w testach podobną wydajność do telefonu Samsung Galaxy S4, który jest od niego droższy ponad trzykrotnie. Widać też znaczny postęp w technologii wytwarzania urządzeń mobilnych gdzie telefony S4 i Core utrzymują lepszą ilość klatek na sekundę, niż tablet iPad 1.

Wykres 3

Jak widać z wykresu 3. przy 50 obiektach ruch tychże nie wpływa znacząco na ilość rysowanych klatek na sekundę. Spadki ponownie pojawiają się przy 250 obiektach na ekranie i są podobne pomiędzy urządzeniami. Wyjątkiem jest tu iPhone 5, który spadek wydajności pokazuje dopiero przy 500 obiektach na ekranie. Warto odnotować fakt iż urządzenia firmy Apple wykazują mniejszą stabilność przy dużej ilości obiektów na ekranie – spadek ilości klatek na sekundę był dużo bardziej wyraźny w ich przypadku.

Wykres 4

Na wykresie 4. zaprezentowano wpływ sztucznych operacji zmiennoprzecinkowych na ilość klatek na sekundę. Symuluje to kod logiki gry uruchomiony obok silnika renderującego. Jak widać większość urządzeń poradziła sobie w teście bardzo dobrze, zauważalne spadki ilości klatek na sekundę wystąpiły tylko na urządzeniach Core i iPad 1.

Wykres 5

Na wykresie 5. zaprezentowano wyniki testu inkrementalnego. Można z niego wywnioskować maksymalne wartości obiektów na ekranie dla poszczególnych urządzeń. Jak widać najgorzej z testowanych urządzeń wraz z ilością obiektów skaluje się telefon Nokia Lumia 625 osiągając niesatysfakcjonujące wyniki już przy ok. 175 obiektach. Najlepiej poradził sobie iPhone 5, który 30 klatek na sekundę rysował dopiero przy ok. 630 obiektach na ekranie. Pozostałe urządzenia poradziły sobie dobrze, pokazując, iż optymalnym dla nich zakresem jest ilość pomiędzy 250 a 500 obiektów.

Wykres 6

Na wykresie 6. przedstawiono wyniki testu tworzenia/destrukcji obiektów. W kontekście wcześniej przeprowadzonych testów maksymalnych ilości obiektów na ekranie widać, iż wartości te są całkiem duże. Sporym zaskoczeniem jest tutaj wydajność telefonu Lumia, który spisał się lepiej niż telefony S4 i Core.

Wykres 7

Jak widać na wykresie 7. praktycznie wszystkie urządzenia uzyskiwały lepsze rezultaty po przejściu w tryb renderowania grafiki 3D. Liderem jest tu telefony typu Core, który dla 250 obiektów na ekranie uzyskał wynik lepszy o ponad 8 klatek na sekundę. Jest to bardzo ciekawy, mimo iż nieoczekiwany wynik. Sugerować może on jeszcze niedokończoną optymalizację modułu 2D w silniku Unity, który to został wprowadzony do pakietu całkiem niedawno, może tez sugerować różnice w shaderach pomiędzy modułami – shadery w wersji 3D rozwijane są od bardzo dawna i przystosowane są do możliwości platform mobilnych zapewniając implementacje alternatywne i tzw. fallback shader, gdzie wersja dwuwymiarowa niekoniecznie. Różnica może też wynikać z braku przełączania tekstur w testach trójwymiarowych, acz różnica ta nie powinna działać na niekorzyść testów dwuwymiarowych do tego stopnia, jako, że tekstury trzymane były w atlasie i zmieniał się tylko ich offset.

Wykres 8

Na wykresie 8. przedstawiono różnice pomiędzy testami dwu a trójwymiarowymi z obiektami ruchomymi. Widać tutaj delikatny spadek wydajności dla telefonów typu S4, Lumia oraz Core. Spadek może wynikać z konieczności przeliczania jednego dodatkowego wymiaru podczas ruchu obiektów (mimo, iż w testach jest on stały), gdzie w testach dwuwymiarowych był od razu ignorowany.

## Unity: Gra 2D

Mając powyższe wyniki testów przystąpiono do tworzenie pierwszej z gier. Jest to prosta gra zręcznościowa podobna do Icy Tower, stworzona w oparciu o przykładowe zasoby dostarczone z silnikiem Unity, takie jak tekstury, muzyka czy tła. 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.



Rysunek 11. Zrzut ekranu z gry zręcznościowej

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.

Podstawowymi komponentami bohatera są obiekty klas RigidBody2D i Collider, których rozkład widać na rysunku 12. 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 12. 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. Obiekt Animator posiada mapę stanów animacji, w których może znajdować się postać. Domyślnym stanem jest stan Idle, w którym postać delikatnie porusza się, co ma wizualizować oddychanie. Ze stanu Idle postać może przejść do stanu Run, pokazującego bieganie, lub Jump, który animuje skok. Te trzy stany są ze sobą połączone relacjami w obu kierunkach, co znaczy, iż z każdego stanu da się przejść w dowolny inny, a samym spajaniem faz animacji zajmuje się silnik Unity.

Tworzeniem platform dla gracza zajmuje się skrypt PlatformCreator, który sprawdza na jakiej wysokości znajduje się gracz oraz na jakiej wysokości znajduje się krawędź kamery, a następnie dbając o to by możliwe było doskoczenie do następnej z platform tworzy kolejną. Skrypt PlatformCreator aktywny jest tylko w pierwszej fazie gry, w pozostałych dwóch zostaje wyłączony. Każda z platform przelicza jak długo istnieje w świecie. Jeżeli przekroczy ustalony czas zniszczenia włączany jest jej element RigidBody2D, który powoduje rozpoczęcie opadania grawitacyjnego danej platformy. Następnie po przekroczeniu krawędzi kamery, gdy platforma jest niewidoczna dla gracza, zostaje zniszczona. Platformy posiadają także skrypt CollisionIgnorerScript, który chwilowo wyłącza obiekty Collider danej platformy pozwalając graczowi wskoczyć z niższej platformy na wyższą. Po opuszczeniu przez gracza wyzwalacza odpowiedzialnego za znajdowanie się gracza pod platformą obiekty Collider zostaną powtórnie włączone i pozwolą graczowi stabilnie stać na kolejnej platformie.

Po zakończeniu fazy wspinania i przejściu w fazę spadania postać gracza otworzy spadochron i wpływ grawitacji zostanie zmniejszony by odzwierciedlić wpływ tegoż. Do akcji wkracza wtedy skrypt FallingEnemyCreator odpowiedzialny za stworzenie obiektów przeciwników, których gracz może „złapać” w drodze ku ziemi. Skrypt ten działa podobnie do skryptu tworzącego platformy, ale narzuca też warunki tworzenia w sposób niepozwalający na przekroczenie maksymalnej ilości obiektów na ekranie dla danego urządzenia. Po złapaniu kilku przeciwników pozostali złapani przeciwnicy zostają zliczani w wewnętrznej zmiennej klasy reprezentującej gracza a ich obiekt zostaje tymczasowo zniszczony w celu zaoszczędzenia cyklów procesora.

Po przejściu do kolejnej, ostatniej fazy, postać gracza zamyka spadochron i uderza o ziemię. Wszyscy zebraniu wtedy przeciwnicy rozlatują się wtedy w kształt stożka poprzez aktywację ich modułów RigidBody2D i nadaniem im siły skierowanej ku górze i w losową stronę. Zliczona zostaje punktacja zdobyta przez gracza i pokazany zostaje mu jego wynik.

## Unity: Curveball

Kolejną z zaimplementowanych aplikacji była trójwymiarowa gra zręcznościowa podobna do Ponga z możliwością gry wieloosobowej jak również przeciwko przeciwnikowi komputerowemu. Widok gracza ma znajdować się za paletką, po przeciwnej stronie w identycznym ustawieniu znajduje się przeciwnik. Piłeczka porusza się w trzech wymiarach pomiędzy oponentami ze zwiększającą się po każdym odbiciu prędkością. Przeciwnicy mogą nadawać piłce spin, który wpływa na jej tor lotu po odbiciu, a także na kąty odbicia po kontakcie ze ścianami. Zdobycie punktu naliczane jest w momencie, gdy piłeczka minie jedną z paletek w sposób uniemożliwiający jej odbicie. W trybie gry jednoosobowej zarówno gracz jak i komputer otrzymują trzy życia, które tracą wraz ze zdobyciem punktu przez przeciwnika. Gdy oponent komputerowy straci wszystkie trzy życia, otrzymuje nowe i poziom jego gry wzrasta. Gdy gracz straci swoje życia gra dobiega końca.

Konstrukcja planszy w grze jest prosta – składa się z 6 ścian ułożonych w prostopadłościan. Wszystkie ze ścian posiadają w sobie element BoxCollider by mogły wchodzić w interakcje z piłką. Dwie najmniejsze ściany mają dodatkowo przyłączony skrypt „scoreField” odpowiedzialny za naliczanie punktów dla odpowiedniego gracza oraz informowanie pozostałych elementów gry o bieżącym wyniku. W momencie inicjalizacji statycznej planszy tworzony jest też niewidzialny obiekt „GameManager” odpowiedzialny za tworzenie odpowiednich menu, monitorowanie pozycji paletek graczy oraz piłki, monitorowanie stanu gry wieloosobowej czy też za stan przepływu czasu w grze. W pierwszej kolejności menedżer gry tworzy pierwsze z widocznych menu, w którym wybrać można opcje gry jednoosobowej, stworzenia serwera lub dołączenia do istniejącego serwera. Po utworzeniu menu menedżer inicjalizuje mecz pokazowy w tle, który składa się z dwojga graczy komputerowych grających między sobą bez limitów punktowych. Co dzieje się dalej zależy już od użytkownika. Zakładając wybór gry jednoosobowej gracz poproszony zostanie o podanie swojego imienia a następnie rozpocznie się gra na opisanych wcześniej zasadach. W wypadku wyboru stworzenia serwera dla gry wieloosobowej gracz proszony jest o podanie swojego imienia a także nazwy swojej gry, następnie czeka na dołączenie się drugiego gracza. Gdy drugi gracz dołączy do gry, czas gry jest wznawiany i oboje grają do momentu osiągnięcia przez jednego z nich 11 punktów. W przypadku wyboru opcji dołączenia do serwera gracz proszony jest o podanie swojego imienia a następnie przedstawiony zostaje mu ekran, na którym może dokonać wyboru gier oczekujących na gościa.

Opis samej implementacji należy zacząć od klasy „GameManager”, a konkretnie od jego funkcji Start(), która jest punktem wejściowym programu:



W funkcji tej w pierwszej kolejności określamy, iż aplikacja nie powinna być zatrzymywana po utracie focusu. Znaczy to, że program będzie wykonywał się dalej po np. zminimalizowaniu jego okna w systemach operacyjnych z rodziny Windows. Przyjęto taki sposób wykonywania, gdyż w przeciwnym wypadku po podłączeniu do sesji wieloosobowej gracz mógłby zminimalizować aplikację i zawiesić wykonywanie gry dla drugiej osoby.

Po ustawieniu opcji „runInBackground” sprawdzany jest warunek toczącej się gry – jeżeli nie trwa aktualnie gra pomiędzy dwojgiem graczy lub pomiędzy graczem i komputerem, włączany jest tryb gry demo pomiędzy dwoma przeciwnikami komputerowymi bez limitów punktowych.

Następnie egzekucja kodu automatycznie przechodzi do funkcji Update(), wykonującej się w każdej klatce gry:



Co klatkę sprawdzany jest warunek rozpoczętej gry i w zależności od jego ewaluacji następuje blokada kursora. Następnie przechwytywany jest przycisk wyjścia w celu pauzowania/wznawiania gry. „Keycode.Escape” nie znaczy stricte klawisza „Esc” na klawiaturze komputera osobistego, w kontekście platform mobilnych znaczy np. przycisk „Back”.



Rysunek 13. Diagram klas aplikacji Curveball

Po zakończeniu tej inicjalizacji przychodzi czas na narysowanie przycisków wyboru gry. Kod obsługujący te operacje znajduje się w klasie „Menu” agregowanej przez klasę GameManager jak pokazano na diagramie widocznym na rysunku 12. Engine Unity automatycznie wywołuje funkcję OnGUI(), w której powinien być umieszczony kod zajmujący się tworzeniem interfejsu. W klasie „Menu” zastosowano mechanizm pozwalający na przełączanie aktywnego menu bez względu na miejsce w kodzie, w którym się znajdujemy. Pozwala to na łatwiejsze interakcje z opcjami, które przedstawiać należy graczom, bez konieczności przekazywania menu do niektórych funkcji. Z klasy „Menu” dziedziczą cztery klasy: SinglePlayerMenu, ServerNameMenu, JoinServerMenu oraz DisconnectMenu. Klasy te nadpisują niektóre metody klasy bazowej w celu np. przesunięcia przycisków w bardziej odpowiednie miejsca, czy też w celu łatwiejszego wywołania niektórych z funkcjonalności aplikacji, takich jak dołączanie do istniejącego serwera czy wybór imienia w trybie jednoosobowym.

Po wybraniu trybu dla jednego gracza i wybraniu imienia gra jest pauzowana a następnie scena jest niszczona – kasowane są paletki, piłka i światła. Nowe obiekty zostają inicjalizowane z tzw. prefabów – gotowych obiektów o odpowiednio skonfigurowanych skryptach i komponentach. W pierwszej kolejności tworzona jest piłka, która jest sferą z dołączonym obiektem klasy SphereCollider, RigidBody oraz skryptem „Ball” odpowiedzialnym za jej zachowanie. Skrypt ten głównie decyduje o maksymalnej prędkości piłki, ale odpowiada także za przemieszczanie obiektów reprezentujących oświetlenie oraz za zmniejszanie kąta odbicia po kontakcie z paletką.

Następnie tworzone są paletki reprezentujące graczy – jedna dla gracza ludzkiego, jedna dla przeciwnika komputerowego. Obie wywodzą się z klasy bazowej „Paddle”, która odpowiada za nakładanie ograniczeń w sposób uniemożliwiający opuszczenie pola gry oraz za zapamiętywanie poprzednich pozycji w wektorze reprezentującym położenie paletki we wcześniejszych klatkach. Wektor ten później wykorzystywany jest do obliczania mocy i kierunku podkręcenia piłeczki po kontakcie z paletką. Paletka typu „PlayerPaddle” dodatkowo posiada mechanizmy przeliczające ruchy myszki (czy też w przypadku platform mobilnych – dotyk ekranu) na jej położenie w przestrzeni trójwymiarowej. Paletka typu „AiPaddle” rózni się tylko sposobem kontroli – nadpisana zostaje metoda „handlePaddleMovement()”. Sposób gry przeciwnika komputerowego jest bardzo prosty, mianowicie zmierza on zawsze w kierunku piłki starając przewidzieć się jej położenie. Różne poziomy trudności przeciwnika polegają jedynie na zmianie zmiennej „dampingFactor” odpowiedzialnej za tłumienie normalnych, szybkich reakcji komputera. Wraz z czasem trwania gry „dampingFactor” wzrasta i przeciwnik komputerowy przemieszcza się szybciej, przez co jego skuteczność znacznie wzrasta.

W razie wybrania przycisku gry wieloosobowej gracz zostanie przeniesiony do ekranu tworzenia gry, w którym należy wpisać swoje imię i nazwę tworzonej gry. Po zatwierdzeniu dokonanych wyborów kod gry zniszczy poprzednią scenę następnie zarejestruje stworzoną grę w głównym serwerze utrzymywanym przez Unity, po czym gra przejdzie w stan oczekiwania na dołączenie się innego gracza. W momencie podłączenia się drugiego gracza skonfiguruje on swoją scenę, a następnie używając RPC (Remote Procedure Call) poinformuje gospodarza o swojej obecności. Instancje gry będą się komunikowały w celu synchronizacji imion graczy a także zresetowania punktacji. W tym momencie stworzone zostaną brakujące elementy sceny, czyli piłka i paletki. W trybie wieloosobowym będą to obiekty typu „MultiplayerPaddle” oraz „MultiplayerBall”. Główną różnica w porównaniu do obiektów tworzonych podczas gry jednoosobowej jest agregacja obiektu typu „NetworkView”, który pozwala na zrzucenie odpowiedzialności za synchronizację obiektów pomiędzy graczami na silnik Unity. Drugą z różnic jest uzależnienie kontroli od wyniku zmiennej „isMine” obiektu „NetworkView”. Zmienna ta decyduje czy konkretny egzemplarz „NetworkView” został stworzony przez obecnego gracza. Należało wprowadzić takie rozgraniczenie by gracze nie interferowali sobie nawzajem w sterowanie paletkami. Gdy sesja gry zostanie zestawiona, aplikacja wznowi wykonanie. Każdy z graczy zapamiętuje swoje poprzednie pozycje i przez to oblicza podkręcenie nadane piłce. Przez sieć synchronizowane pomiędzy graczami są tylko pozycja piłki oraz paletek. Odbywa się to samoczynnie, przez wymieniony wcześniej obiekt „NetworkView”.



Rysunek 14. Zrzut ekranu z gry trójwymiarowej

# Podsumowanie i wnioski

W niniejszej pracy pokazano, iż stworzenie interaktywnej, wieloplatformowej aplikacji do wizualizacji obiektów dwu i trójwymiarowych jest osiągalne. Ukazano sposób konfiguracji środowisk oraz stworzenia aplikacji na najpopularniejsze platformy mobilne przy użyciu silnika Unity. Stworzone aplikacje posiadają wspólny kod oraz zasoby i mogą być uruchomione na systemach operacyjnych z rodziny iOS, Android czy Windows Phone 8 bez modyfikacji. Takie podejście pozwala na znaczne oszczędności czasu programistów, co przekłada się na namacalne oszczędności w zasobach finansowych. Silnik Unity okazał się bardzo dobrym narzędziem zarówno pod względem przenośności kodu, jak i samego projektowania aplikacji, pomimo wielu wad wbudowanego edytora MonoDevelop. Projektowanie aplikacji w Unity wymaga trochę innego podejścia niż projektowanie tradycyjnych programów. W celu łatwiejszego testowania aplikacje powinny być modularne, co najlepiej osiąga się poprzez wykorzystanie wbudowanego w silnik systemu scen i obiektów prefab, pozwalających na tworzenie gotowych i skonfigurowanych instancji klas wraz z podpiętymi zachowaniami i atrybutami.

Najbardziej uciążliwymi częściami samego procesu była konfiguracja środowiska, szczególnie dla kompilacji na system iOS, który sprawiał najwięcej problemów z powodu konieczności spełnienia warunków licencyjnych. Pomimo, iż zaproponowane w pracy rozwiązania pozwalają na kompilację na wszystkie testowane platformy najmniejszym możliwym kosztem w zgodzie z licencjami wszystkich firm, proces ten jest bardzo uciążliwy i czasochłonny, i raczej nie sprawdziłby się w codziennej pracy. W celu uniknięcia podobnych nieprzyjemności w przyszłości należałoby proces produkcji aplikacji przenieść na urządzenie z systemem Mac OS, w celu posiadania możliwości kompilacji na iOS, a następnie zainstalować system Windows 8 w maszynie wirtualnej korzystając np. z wymienionej aplikacji Virtualbox, by mieć możliwość kompilowania na urządzenia z systemem Windows 8.

Z zaimplementowanych testów wynika, iż pomimo znacznego postępu w dziedzinie urządzeń przenośnych ich wydajność pozostawia jeszcze wiele do życzenia i optymalizacja kodu jest dalej bardzo ważnym zagadnieniem. Wyniki pokazały, iż większość współczesnych urządzeń potrafi poradzić sobie bez większych spadków wydajności renderowania przy 50 obiektach na ekranie, a problemy pojawiają się dopiero przy 250 obiektach. Bardzo widoczne różnice w wydajności aplikacji widać przy wprowadzeniu ruchów obiektów – aplikacje spowalniały średnio o ok. 10 klatek na sekundę po wprowadzeniu do testu ruchomych obiektów. Wpływ operacji zmiennoprzecinkowych wykonywanych w każdej klatce obliczeń był bardzo znikomy. Nieoczekiwanym wynikiem był wzrost wydajności na urządzeniach po przejściu wykonywania testów nieruchomych do trybu trójwymiarowego. Wynikać może to z jeszcze młodej, niezoptymalizowanej implementacji modułu dwuwymiarowego w Unity. Z testu możliwości tworzenia/niszczenia obiektów wywnioskowano, iż urządzenia tej samej generacji są raczej zbliżone możliwościami w tym aspekcie i że przy obecnych ograniczeniach urządzeń mobilnych nie jest problemem szybkie tworzenie wielu obiektów, jako że po chwili i tak osiągnęlibyśmy maksymalną możliwą ilość obiektów na ekranie bez utraty płynności w aplikacji. W celu polepszenia doznań użytkownika naszych aplikacji możliwym byłoby uruchamianie podobnych testów w trakcie instalacji lub pierwszego uruchomienia aplikacji by sprawdzić możliwości danego urządzenia, a następnie dostosowanie ilości obiektów w grze do wyników analizy.

Możliwymi rozwinięciami niniejszej pracy mogłoby być przeprowadzenie testów na większej ilości urządzeń i stworzenie swoistej ustandaryzowanej bazy możliwości urządzeń. Istniejące rozwiązania, pomimo iż dają możliwość porównania urządzeń pod względem wydajności raczej nie pozwalają na porównanie możliwości np. modułu graficznego, przedstawiając wyniki, jako całościową wydajność urządzenia. Baza z bardziej szczegółowymi wynikami dla każdego z urządzeń mogłaby przedstawiać znaczną wartość dla osób tworzących aplikacje dla danego środowiska. Stworzone testy można też rozwinąć o kolejne. Należałoby sprawdzić maksymalny ruch sieciowy pomiędzy urządzeniem a serwerem, prędkości pamięci podręcznej czy prędkość dostępu do pamięci wewnętrznej i zewnętrznej danego urządzenia.

# Literatura

1. A. Gupta, R. Cozza, CK Lu, *Market Share Analysis: Mobile Phones, Worldwide, 4Q13 and 2013*, Stanford 2014, s. 6 [↑](#footnote-ref-1)