Zrealizuj swój pomysł na grę i i zdobądź uznanie Użytkowników!

Dotknij, Przesuń, Potrząśnij

Od pomysłu do gry na Phone'a i iPada

Helion
O'REILLY

Todd Moore

Tytuł oryginału: Tap, Move, Shake: A Hands-on Guide to Creating Multi-touch Games with iPad and iPhone

Tłumaczenie: Robert Górczyński

ISBN: 978-83-246-3965-6

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/dotkni.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/dotkni
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

O autorze **7**

Wstęp **9**

Wprowadzenie **11**

Dla kogo przeznaczona jest ta książka? 12
Czego się nauczysz? 13
Konwencje zastosowane w książce 14
Użycie przykładowych kodów 15

1. Wprowadzenie do Xcode **16**

Rejestracja konta programisty 17
Instalacja Xcode 17
Xcode 19
Moduł Interface Builder 33
Połączenia 36
Logika gry 39

2. Hello Pong **44**

Utworzenie projektu 45
Interfejs użytkownika gry 48
Wyświetlacz Multi-Touch 52
Animacja 59
Wykrywanie kolizji 62
Punktacja 63
Wykończenie gry 66
Dźwięk 74

3. Grafika 80
Wprowadzenie 82
Grafika rastrowa i wektorowa 83
Formaty obrazów 84
Wyświetlacz Retina 85
Utworzenie obrazów dla gry Air Hockey 87
Integracja aplikacji 102
Kompilacja i uruchomienie gry 106

4. Fizyka 108
Fizyka podczas ruchu paletek 109
Fizyka podczas ruchu krążka 120

5. Dźwięk 134
Czym jest dźwięk? 135
Tworzenie dźwięków 138
Pobieranie dźwięków 138
Nagrywanie dźwięku 139
Edycja dźwięku 141

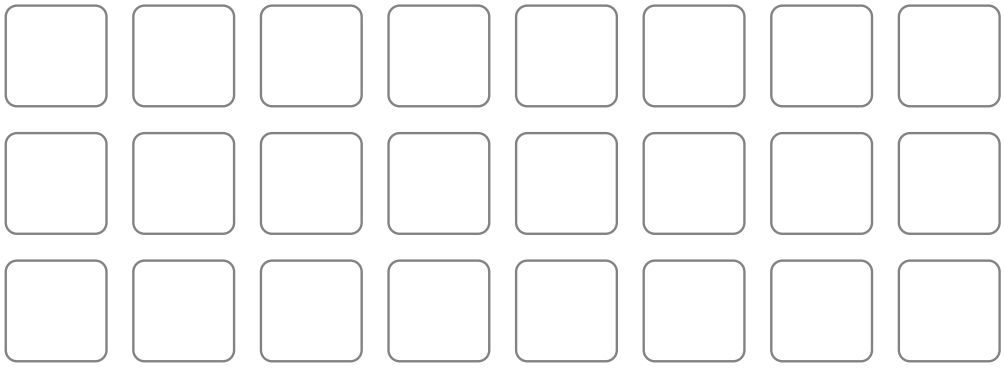
6. Sztuczna inteligencja 146
Przygotowanie menu głównego 147
Gracz-komputer 154
Poziom trudności 169

7. App Store 176
Zrzuty ekranu 177
Przygotowanie opisu aplikacji i wybór słów kluczowych 179
Podanie metadanych w iTunes Connect 180
Archiwizacja i wysłanie aplikacji 184
Recenzja aplikacji 187
Marketing i sprzedaż 193
Podsumowanie 201

Skorowidz 203

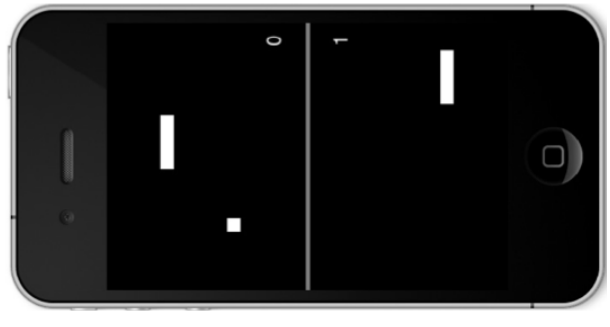
2.

Hello Pong



Moje uzależnienie od gier wideo rozpoczęło się z chwilą, gdy tata kupił konsolę Atari Home Pong. Konsola była podłączana do telewizora, posiadała dwa kontrolery i jedynie czarno-białą grafikę. Po obu stronach ekranu były wyświetlane dwa prostokąty oznaczające paletki sterowane przez graczy. Kontroler był wyposażony w obrotową tarczę pozwalającą na poruszanie paletką w pionie. Na ekranie była wyświetlana także piłeczka w kształcie białego kwadratu, która odbijała się od ścian oraz paletek sterowanych przez graczy. Każde kolejne uderzenie piłki zwiększało jej prędkość, co jeszcze bardziej utrudniało jej kolejne odbicie paletką. Jeżeli graczowi nie udało się odbić piłeczki, przeciwnik otrzymywał punkt, następowało zakończenie danej rundy i przywrócenie piłeczce jej prędkości początkowej.

Wprawdzie gra Pong nie miała przyciągającej uwagi grafiki i dźwięku, to jednak charakteryzowała się wszystkimi typowymi elementami aktualnie wydawanych gier, takimi jak cel do zrealizowania, udział graczy, punkty wskazujące na uzyskany postęp w grze oraz sposób ukończenia gry. Inżynier Atari, który zaprojektował i zbudował grę Pong, otrzymał zadanie wykonania tego projektu jako ćwiczenia mającego mu pomóc w przygotowaniach do tworzenia gier. Uważam, że takie ćwiczenie nawet w obecnych czasach jest doskonałym sposobem poznania metody tworzenia gier dla urządzeń iPhone (zobacz rysunek 2.1). Podczas pracy nad tym projektem dowiesz się, jak zaimplementować obsługę wyświetlacza Multi-Touch, animacji, wykrywania zderzeń oraz punktacji.

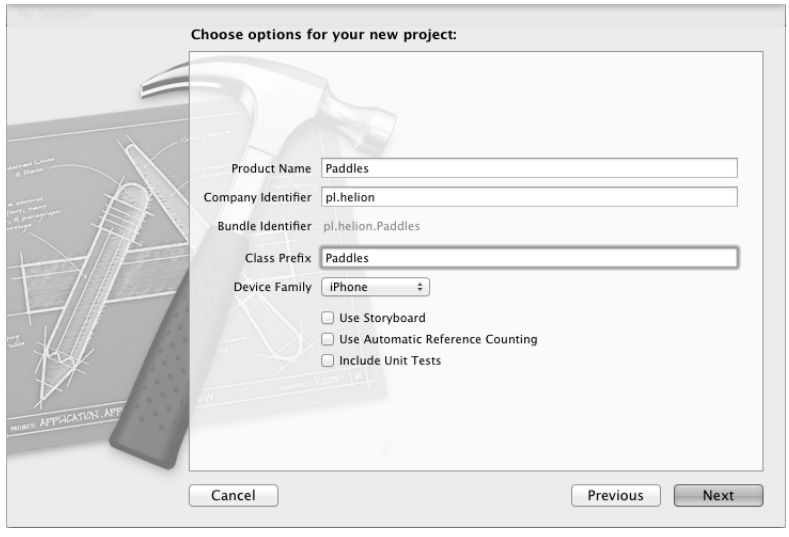


Rysunek 2.1. Gra Paddles uruchomiona w symulatorze iPhone

Utworzenie projektu

Pierwszym krokiem jest uruchomienie Xcode, utworzenie nowego projektu (*File/New/New Project*) i nadanie mu nazwy *Paddles*. Wybierz szablon *iOS/Applications/Single View Application* i kliknij przycisk *Next*. Podaj *Paddles* jako nazwę produktu i prefiks klas, podaj odpowiedni

identyfikator firmy, jako urządzenie wybierz iPhone i usuń zaznaczenie z trzech pól wyboru znajdujących się na dole (zobacz rysunek 2.2). Kliknij przycisk *Next*, wybierz miejsce zapisu projektu, a następnie kliknij przycisk *Create*.



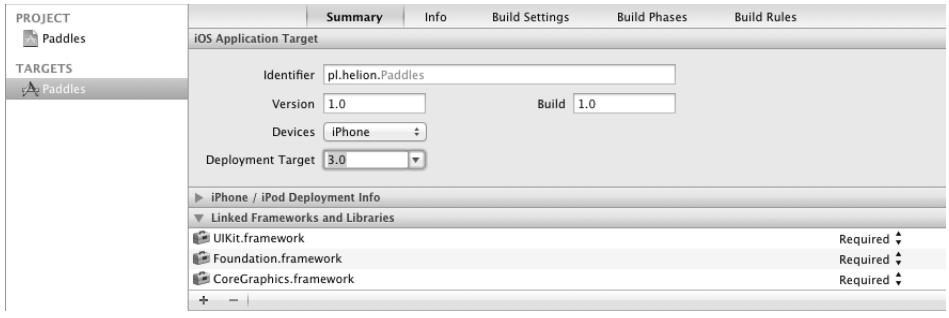
Rysunek 2.2. Utworzenie nowego projektu: Paddles

Po wygenerowaniu nowego projektu przez Xcode wyświetlana jest strona ustawień, na której trzeba będzie wprowadzić kilka zmian. Wszystko zostało opisane poniżej.

Ustawienia aplikacji

Ekran Summary dla aplikacji Paddles zawiera kilka ważnych elementów, między innymi obsługiwane urządzenia oraz minimalną wymaganą wersję systemu iOS. Rozwijane menu *Devices* pozwala na wybór urządzenia docelowego: *iPhone*, *iPad* lub *Universal* (obsługa obydwu wymienionych urządzeń za pomocą pojedynczej aplikacji). W tworzonym projekcie pozostawiamy *iPhone* jako urządzenie docelowe, ale minimalną wymaganą wersję systemu iOS ustalamy na 3.0 (zobacz rysunek 2.3). Zawsze warto ustalać jak najmniejszą wymaganą wersję systemu iOS do uruchomienia aplikacji, aby nie eliminować potencjalnych klientów, którzy nie posiadają najnowszych urządzeń iOS. Budowana w tym rozdziale gra nie wymaga funkcji znajdujących się jedynie w najnowszych wydaniach systemu iOS.

Aplikacje wykorzystujące funkcje, które wymagają nowszej wersji systemu iOS niż zdefiniowana w ustawieniach, ulegną awarii po uruchomieniu ich w urządzeniach działających ze starszą wersją systemu niż wymagana. Bardzo ważne jest więc sprawdzenie wersji systemu iOS podczas uruchamiania aplikacji lub wcześniejsze zdefiniowanie minimalnej wymaganej przez funkcje użyte w danej aplikacji. W dokumentacji iOS zawsze należy sprawdzać wersję systemu wymaganą dla danej funkcji.



Rysunek 2.3. Ustawienia aplikacji oraz zdefiniowanie minimalnej wymaganej wersji systemu iOS

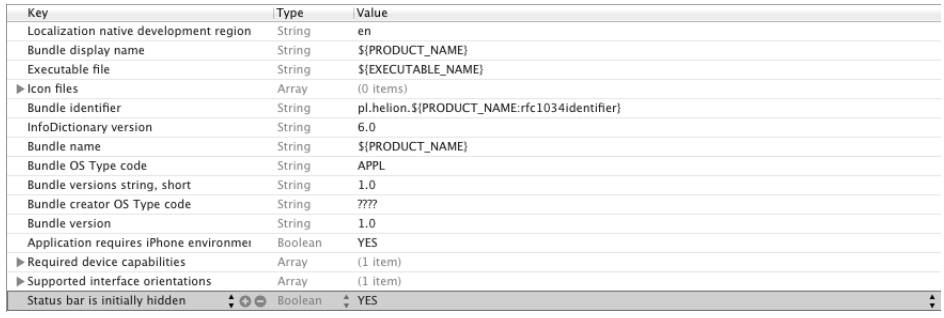
Istnieje również możliwość podania układów ekranu obsługiwanych przez aplikację — w sekcji *Supported Devices Orientations* dostępne są opcje *Portrait*, *Portrait Upside Down*, *Landscape Left* oraz *Landscape Right*. Ma to szczególne znaczenie w przypadku aplikacji dla tabletu iPad, ponieważ mogą być one uruchamiane w dowolnym położeniu urządzenia (stąd konieczność zapewnienia odpowiedniego ekranu powitalnego dla położenia poziomego i pionowego). Jednak w przyszłości może to ulec zmianie w przypadku smartfona iPhone, więc najlepszym rozwiązaniem jest poinformowanie systemu iOS, które układy są obsługiwane przez aplikację. Gra Paddles obsługuje jedynie układ pionowy, upewnij się więc, że zaznaczony jest jedynie przycisk *Portrait* w sekcji *Supported Devices Orientations*.

Kolejnym krokiem jest wprowadzenie kilku zmian w pliku typu App Info.

> Klasa kontrolera widoku także określa obsługiwane układy ekranu. Domyślna implementacja zapewnia obsługę jedynie układu pionowego (*Portrait*), ale wygenerowany kod projektu może zapewnić obsługę pozostałych układów. Otwórz plik *PaddlesViewController.m* i usuń metodę `shouldAutorotateToInterfaceOrientation`, o ile taka istnieje. W ten sposób gwarantujesz, że kontroler widoku nie będzie zmieniał układu ekranu podczas gry.

Plik typu App Info

Gry w smartfonie iPhone najczęściej wykorzystują całą dostępną powierzchnię ekranu, co oznacza ukrycie paska stanu. W pliku App Info znajduje się wiele informacji opisujących aplikację iOS, między innymi wersja, pliki ikon oraz wyświetlana nazwa. Wymienione informacje można edytować poprzez rozwinięcie grupy *Supporting Files* i otworzenie pliku *Paddles-Info.plist* albo poprzez kliknięcie karty *Info* na ekranie Paddles Target. Wiele ustawień nie jest domyślnie wyświetlanych, a możliwość ukrycia paska stanu zalicza się do jednej z nich. Musisz więc dodać nowy element, klikając jeden z istniejących elementów, a następnie ikonę plusa lub po prostu klikając prawym przyciskiem myszy i wybierając opcję *Add Row* z menu kontekstowego. W polu *Key* nowo dodanego elementu z rozwijanego menu wybierz opcję *Status bar is initially hidden* i ustaw jej wartość `YES`, jak pokazano na rysunku 2.4. Powodem dodania tej opcji w tym pliku zamiast w kodzie aplikacji jest to, że system powoli ukryje pasek stanu podczas wczytywania aplikacji. Animowane ukrycie paska stanu podczas wczytywania aplikacji wygląda znacznie lepiej niż po prostu jego natychmiastowe ukrycie tuż po wczytaniu aplikacji.



Rysunek 2.4. Ukrycie paska stanu w aplikacji iOS

Interfejs użytkownika gry

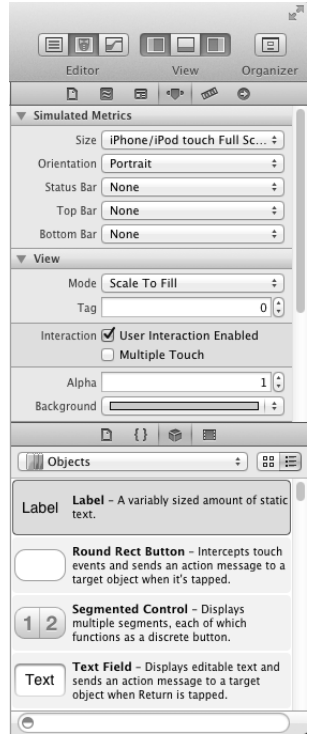
Pierwszą grę dla smartfona iPhone utworzyłem bez używania programu Interface Builder. Oznacza to, że musiałem ręcznie zaalokować wszystkie widoki, obrazy i etykiety poprzez odpowiednie dobranie ich wielkości i położenia w taki sposób, aby wszystko przedstawiało się dobrze. Wszystko, co mogłem szybko zrobić w programie Interface Builder, tworzyłem powoli w kodzie źródłowym, co jest wyjątkowo żmudne i podatne na powstawanie błędów. Gdybym miał możliwość cofnięcia się w czasie i poświęcenia pół godziny na poznanie sposobu używania edytora WYSIWYG (ang. *What You See Is What You Get*, czyli otrzymasz to, co widzisz na ekranie) Interface Builder, grę ukończyłbym znacznie wcześniej.

Moduł Interface Builder

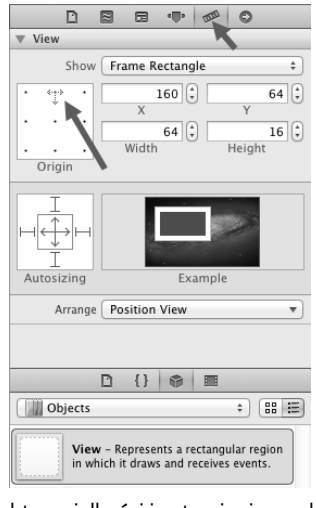
W panelu nawigacyjnym kliknij plik *PaddlesViewController.xib*, w obszarze edytora Xcode zostanie wyświetlony moduł Interface Builder pozwalający na utworzenie bądź zmodyfikowanie interfejsu użytkownika. Następnie upewnij się o wyświetleniu inspektora atrybutów w panelu narzędziowym (wybierz opcję menu *View/Utilities/Attributes Inspector*). Panel narzędziowy jest wyświetlany po prawej stronie okna Xcode i pozwala na wybór jednego z kilku inspektorów za pomocą paska narzędziowego znajdującego się na górze panelu. Na rysunku 2.5 pokazano inspektora atrybutów (na górze) oraz bibliotekę obiektów (na dole). Kliknij jedyny widok wyświetlony na środku ekranu modułu Interface Builder. Teraz dowiesz się, jak za pomocą inspektora atrybutów zmodyfikować różne właściwości tego *widoku głównego* (ang. *root view*).

> To dobra chwila na poznanie pozostałych inspektorów dostępnych w panelu narzędziowym. Umieszczaj kursor myszy nad ikonami poszczególnych inspektorów, co spowoduje wyświetlenie ich nazw.

Zmień kolor tła (ang. *Background*) widoku z szarego na czarny (ang. *Black*). W rozwijanym menu *Status Bar* w sekcji *Simulated Metrics* wybierz opcję *None*, ponieważ pasek stanu nie będzie wyświetlany w grze. Pasek stanu zabiera na górze ekranu 20 pikseli z całkowitej ich liczby 480. Tak więc przy wyświetlonym pasku stanu widok główny ma 460 pikseli wysokości. Przejdź do inspektora wielkości (zobacz rysunek 2.6) i upewnij się, że wysokość jest ustawiona na 480 pikseli, co odpowiada wysokości ekranu bez paska stanu.



Rysunek 2.5. Panel narzędziowy, inspektor atrybutów i biblioteka obiektów



Rysunek 2.6. Inspektor wielkości i ustawienie punktu początkowego

Biblioteka obiektów znajduje się w dolnej części panelu narzędziowego, tuż pod inspektorami, i pozwala na przeciąganie nowych elementów interfejsu użytkownika do widoku. W bibliotece można znaleźć mnóstwo elementów, między innymi przyciski, etykiety itd. Rozpoczniemy od użycia podstawowych elementów: widoku (ang. *View*) oraz obiektu, po którym dziedziczą wszystkie elementy interfejsu użytkownika. Obiekt `View` posiada

właściwości takie jak wymiary ramki i kolor tła. To wszystko, czego tak naprawdę w tym momencie potrzebujemy, ponieważ paletki i piłeczka będą w grze przedstawiane za pomocą białych prostokątów.

Przewijaj listę elementów biblioteki, aż znajdziesz obiekt `View`, a następnie przeciągnij go do widoku głównego, tworząc w ten sposób podwidok dla istniejącego widoku głównego. Wymiary podwidoku to 64 piksele szerokości i 16 pikseli wysokości; ten widok będzie przedstawiał jedną z paletek. Paletka ma zostać umieszczona na górze, ale powinieneś pozostawić nieco miejsca na umieszczenie palca za paletką.

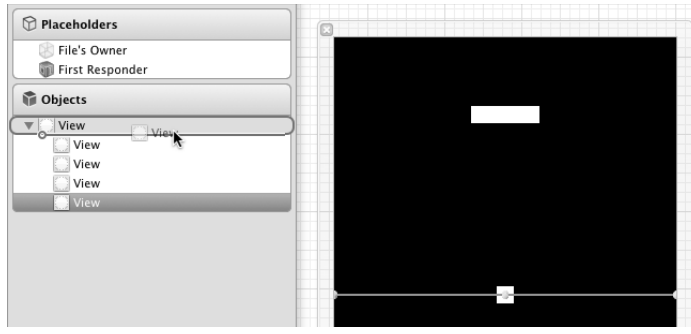
Kliknij obszar *Origin* i zmień położenie punktu początkowego na wyśrodkowany na górze (zobacz strzałkę po lewej stronie rysunku 2.6), co spowoduje dopasowanie położenia punktu początkowego do położenia paletki względem zawierającego ją widoku. Pozycja X paletki powinna być wyśrodkowana w pozycji 160 pikseli od lewej strony ekranu, natomiast pozycja Y paletki to 64 piksele od góry ekranu.

Skopiuj i wklej tę paletkę, tworząc w ten sposób drugą o takich samych wymiarach. Druga paletka powinna znajdować się w takiej samej odległości od dolnej krawędzi ekranu jak pierwsza od górnej. Całkowita wysokość widoku wynosi 480 pikseli, więc trzeba wziąć pierwszy widoczny piksel (479) i odjąć od tej wartości 64 (479 – 64 = 415). Punkt początkowy drugiej paletki powinien być położony na środku dolnej części ekranu, w położeniu 160, 415. Obydwie paletki powinny być teraz wyśrodkowane i znajdować się w takiej samej odległości od krawędzi ekranu.

Kolejnym krokiem jest utworzenie piłeczki poprzez przeciągnięcie nowego widoku i zmianę jego wielkości na 16×16 pikseli. Następnie kliknij rozwijane menu *Arrange Position View* i wybierz opcję *Center Horizontally In The Container*. Teraz ponownie kliknij rozwijane menu *Arrange Position View* i wybierz opcję *Center Vertically In The Container*. Po tych zabiegach piłeczka powinna znajdować się dokładnie na środku widoku.

Na środku ekranu dodałem poziomą linię dzielącą ekran na pół i pomagającą w wizualnym oddzieleniu stron ekranu poszczególnych graczy. Przeciągnij więc kolejny widok i zmień jego wymiary na 320×5 pikseli. Widok powinien zostać wyśrodkowany pionowo i poziomo, podobnie jak w przypadku piłeczki. Przejdź do inspektora atrybutów i zmień kolor tła na szary (ang. *Grey*). Zwróć uwagę, że linia została położona na piłeczce, ponieważ przedstawiający ją widok został dodany jako ostatni.

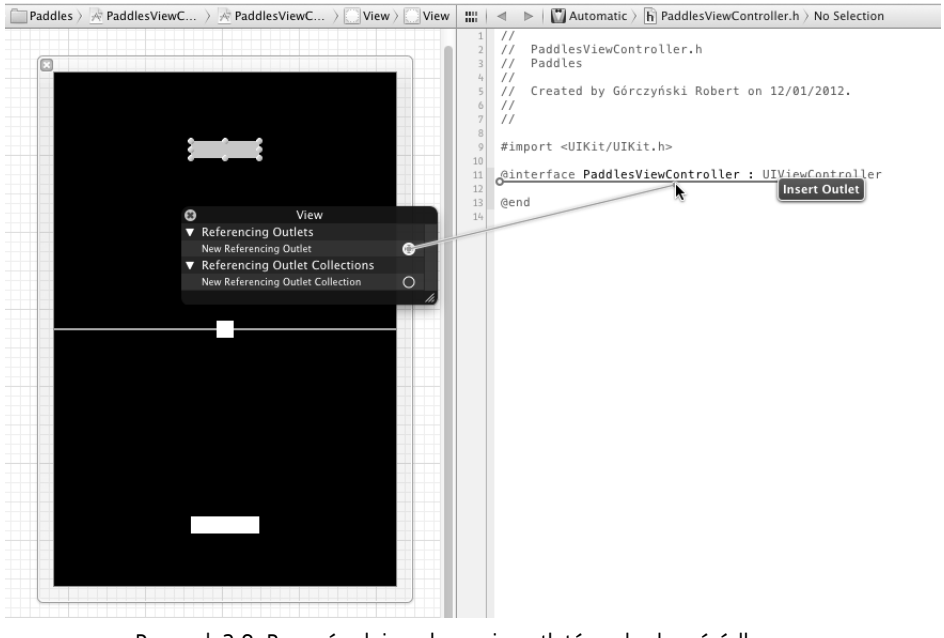
Kolejność obiektów możesz zmienić, używając panelu *Outline View* położonego przy lewej krawędzi edytora. Jeśli panel jest niewidoczny, kliknij jego ikonę wyświetloną w lewym dolnym rogu obszaru edytora. Panel Outline View wyświetla hierarchiczne drzewo odzwierciedlające związki nadrzędny – potomny pomiędzy obiektami w pliku nib. Obiekty znajdujące się na dole drzewa są wyświetlane nad umieszczonymi w górnej części drzewa. Przeciągnij widok linii dzielącej ekran, aby stał się pierwszym w hierarchii obiektów (zobacz rysunek 2.7). W ten sposób linia dzieląca ekran będzie wyświetlana pod piłeczką. Tę samą operację można wykonać, wybierając opcję menu *Editor/Arrangement/Send to Back*.



Rysunek 2.7. Zmiana hierarchii widoków

Połączenia

Podobnie jak w poprzednim rozdziale, także tutaj trzeba wyświetlić panel drugiego edytora, aby plik *PaddlesViewController.h* został wyświetlony obok panelu edytora Interface Builder. Przytrzymując wciśnięty klawisz *Control*, przeciągnij kursor myszy od górnej paletki do wiersza znajdującego się pod definicją `UIViewController`. Podczas przeciągania kursora zobaczysz wyświetlaną na ekranie linię (zobacz rysunek 2.8), jak również zaznaczone miejsce umieszczenia outletu.



Rysunek 2.8. Bezpośrednie połączenie outletów z kodem źródłowym powoduje automatyczne utworzenie właściwości

Po zwolnieniu przycisku myszy zobaczysz menu kontekstowe, które pozwala na określenie typu połączenia, nazwy oraz typu obiektu. Tworzone tutaj połączenie powinno być outletem, nazwa obiektu to `viewPaddle1`, natomiast jego typ to `UIView`. Powtórz ten

proces dla drugiej paletki i piłeczki. Po utworzeniu połączeń plik nagłówkowy powinien mieć postać:

```
@interface PaddlesViewController : UIViewController

@property (nonatomic, retain) IBOutlet UIView *viewPaddle1;
@property (nonatomic, retain) IBOutlet UIView *viewPaddle2;
@property (nonatomic, retain) IBOutlet UIView *viewPuck;

@end
```

Przejdź do pliku implementacji *PaddlesViewController.m* i zwróć uwagę, że podobnie jak w poprzednim rozdziale, także tutaj widoki zostały usunięte z pamięci w metodach `dealloc` i `viewDidUnload`:

```
- (void)dealloc
{
   [viewPaddle1 release];
   [viewPaddle2 release];
   [viewPuck release];
   [super dealloc];
}

- (void)viewDidUnload
{
   [self setViewPaddle1:nil];
   [self setViewPaddle2:nil];
   [self setViewPuck:nil];
   [super viewDidUnload];
}
```

Poprzez połączenie obydwu paletek i piłeczki jako właściwości kontrolera widoku uzyskasz dostęp do ich położenia na ekranie i będziesz mógł nimi manipulować, gdy zajdzie potrzeba. Kolejnym krokiem jest dodanie obsługi paletek za pomocą wyświetlacza Multi-Touch.

Wyświetlacz Multi-Touch

Wprowadzenie przez firmę Apple na rynek jej pierwszego smartfona iPhone pokazało światu imponującą listę innowacji zastosowanych w tak małym urządzeniu. Wyświetlacz Multi-Touch na pewno znajdował się na początku tej listy. Choć ekrany dotykowe były dostępne już od pewnego czasu, to firma Apple pokazała światu, jak tę technologię można efektywnie wykorzystać w produkcie konsumenckim. Cały interfejs użytkownika oraz system operacyjny zostały zbudowane wokół koncepcji dotyku i wyświetlacza Multi-Touch. Interfejs został zaprojektowany jako intuicyjny i łatwy w użyciu, co niewątpliwie przyczyniło się do zyskania ogromnej popularności przez smartfona iPhone. Wcześniej próbowano już dodać obsługę dotyku do tradycyjnych biurkowych systemów operacyjnych, ale te próby najczęściej sprowadzały się do mapowania punktu ekranu dotkniętego przez użytkownika na położenie kursora myszy. Takie rozwiązanie nie działa tak samo jak zastosowane w urządzeniach iOS, które od początku zostały zaprojektowane z myślą o sterowaniu nimi i kontrolowaniu ich za pomocą dotyku.

W pierwszej chwili możesz porównać dotyk do obsługi zdarzeń myszy, ale pomiędzy wymienionymi rozwiązaniami istnieje wiele różnic. Pierwszą i najbardziej oczywistą jest wiele położeń palców na ekranie w tym samym czasie. Pierwszy smartfon iPhone obsługiwał śledzenie do pięciu położeń w tym samym czasie. Druga różnica, która nie musi być aż tak oczywista, to fakt, że czasami ekran w ogóle nie jest dotykany przez użytkownika. Jeżeli nie dotykasz ekranu, to nie ma położenia dotyku. Porównaj to do myszy, której kursor zawsze znajduje się na ekranie. Nawet jeśli nie dotykasz myszy, jej kursor pozostaje na ekranie, a system może sprawdzić jego położenie. Z wymienionych powodów działanie zdarzeń dotyku zostanie tutaj dokładnie omówione. Ponadto zapoznasz się z najlepszymi praktykami stosowanymi podczas śledzenia wielu dotknięć ekranu.

Cztery metody dotyku

Wyświetlacz Multi-Touch jest obsługiwany poprzez dodanie czterech metod do obiektu kontrolera widoku. System będzie wywoływał te metody za każdym razem, gdy nastąpi zmiana stanu dotyku. Metoda `touchesBegan` jest wywoływana po wykryciu pierwszego położenia palca na ekranie. Po przesunięciu palca w nowe położenie nastąpi wywołanie metody `touchesMoved`. Wreszcie po podniesieniu palca z ekranu wywołana będzie metoda `touchesEnded`. Istnieje możliwość, że metoda `touchesEnded` nie zostanie wywołana, jeśli system zdecyduje o przerwaniu obsługi dotknięcia. W takim przypadku zamiast wywołania wymienionej następuje wywołanie metody `touchesCancelled`. Przerwanie może wystąpić w przypadku zakłócenia działania aplikacji poprzez inną funkcję urządzenia, na przykład po otrzymaniu wiadomości lub połączenia przychodzącego.

W kontrolerze widoku zaimplementujemy teraz kod powodujący wyświetlenie w oknie konsoli modułu usuwania błędów nazwy metody aktualnie obsługującej dotyk. W pliku *PaddlesViewController.m* umieść przedstawiony poniżej kod:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
   NSLog(@"touchesBegan");
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
   NSLog(@"touchesMoved");
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
   NSLog(@"touchesEnded");
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
   NSLog(@"touchesCancelled");
}
```

Upewnij się o wyświetleniu okna konsoli w Xcode poprzez włączenie panelu modułu usuwania błędów za pomocą paska narzędziowego lub wybór opcji menu *View/Debug Area/Show Debug Area*. Uruchom aplikację w symulatorze i szybko kliknij na ekranie. Dane wyjściowe w oknie konsoli powinny być podobne do przedstawionych poniżej:

```
2011-03-23 12:03:28.791 Paddles[6007:207] touchesBegan
2011-03-23 12:03:28.990 Paddles[6007:207] touchesEnded
```

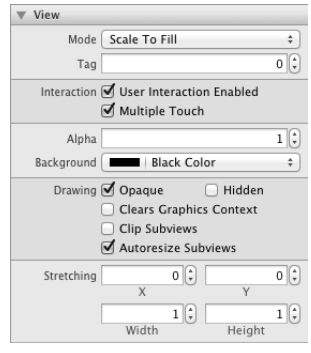
Zwróć uwagę, że nie wszystkie zdarzenia dotyku muszą zostać wygenerowane podczas obsługi dotyku. Jednak zawsze będzie zdarzenie `touchesBegan` oraz `touchesEnded` lub `touchesCancelled`. Teraz kliknij i przeciągnij kursor myszy po ekranie symulatora iOS. Skutkiem będzie wygenerowanie wielu zdarzeń zmiany położenia palca na ekranie pomiędzy zdarzeniami `touchesBegan` i `touchesEnded`.

```
2011-03-23 12:04:09.025 Paddles[6007:207] touchesBegan
2011-03-23 12:04:10.884 Paddles[6007:207] touchesMoved
2011-03-23 12:04:10.933 Paddles[6007:207] touchesMoved
2011-03-23 12:04:11.066 Paddles[6007:207] touchesMoved
2011-03-23 12:04:11.766 Paddles[6007:207] touchesEnded
```

Włączenie obsługi wyświetlacza Multi-Touch

Na tym etapie używamy symulatora iOS do monitorowania wszystkich zdarzeń dotyku ekranu. Symulator ma możliwość emulacji dwóch jednoczesnych dotknięć poprzez przytrzymanie klawisza *Option* podczas klikania. To jednak bardzo ograniczona możliwość, najlepiej dopasowana do obsługi gestu rozciągania w celu zwiększenia poziomu powiększenia. Z tego powodu podczas implementacji obsługi wyświetlacza Multi-Touch najlepiej korzystać z rzeczywistego urządzenia iOS. Podłącz urządzenie iOS, a następnie schemat aktywnego projektu zmień na *iOS Device*.

Jeżeli skompilujesz i uruchomisz aplikację w urządzeniu, a następnie położysz dwa palce na ekranie, to zauważysz, że drugie dotknięcie zostało zignorowane. Wynika to z faktu domyślnego ignorowania wielu dotknięć przez widok. Tę możliwość musisz więc ręcznie włączyć dla każdego wymagającego jej widoku. Odbywa się to poprzez modyfikację właściwości `multipleTouchEnabled` widoku głównego lub użycie modułu Interface Builder i włączenie Multiple Touch, jak pokazano na rysunku 2.9.



Rysunek 2.9. Użycie modułu Interface Builder do włączenia obsługi wielu dotknięć

Metody obsługi dotknięć w kontrolerze widoku będą teraz wywoływane dla każdego dotknięcia na ekranie. Trzeba koniecznie zapamiętać, że każdy obiekt `UITouch` ma gwarancję pozostania tym samym egzemplarzem przez cały cykl życiowy aplikacji, od początku do końca. W ten sposób każde poszczególne dotknięcie ekranu jest przedstawiane za pomocą tego samego obiektu `UITouch` we wszystkich wywołaniach. Aby się o tym przekonać,

dodaj poniższy fragment kodu do zaimplementowanych wcześniej wszystkich metod obsługi dotyku:

```
for (UITouch *touch in touches)
{
   NSLog(@" - %p", touch);
}
```

Powyższy kod powoduje wyświetlenie adresu w pamięci każdego obiektu `UITouch` w zbiorze. Po uruchomieniu aplikacji w rzeczywistym urządzeniu iOS i umieszczeniu dwóch palców na ekranie wygenerowane dane wyjściowe w konsoli będą podobne do przedstawionych poniżej:

```
2011-03-23 14:48:05.015 Paddles[2962:307] touchesBegan
2011-03-23 14:48:05.019 Paddles[2962:307]  - 0x12eed0
2011-03-23 14:48:05.021 Paddles[2962:307]  - 0x12f3b0
2011-03-23 14:48:05.077 Paddles[2962:307] touchesMoved
2011-03-23 14:48:05.080 Paddles[2962:307]  - 0x12eed0
2011-03-23 14:48:05.083 Paddles[2962:307] touchesEnded
2011-03-23 14:48:05.086 Paddles[2962:307]  - 0x12f3b0
2011-03-23 14:48:05.093 Paddles[2962:307] touchesEnded
2011-03-23 14:48:05.096 Paddles[2962:307]  - 0x12eed0
```

Zwróć uwagę, że obydwa przykładowe dotknięcia zostały zarejestrowane w tym samym czasie w metodzie `touchesBegan` i wskazują adresy `0x12eed0` i `0x12f3b0`. Dotknięcie obsługiwane przez obiekt `0x12eed0` zmienia położenie, podczas gdy drugie nie. Wiadomo, że drugie dotknięcie nie zmieniło położenia, ponieważ nie stanowi części zbioru. Dotknięcie obsługiwane przez obiekt znajdujący się pod adresem `0x12f3b0` przechodzi do stanu zakończonego, podobnie jak obsługiwane przez obiekt `0x12eed0` tuż po nim. Na tym etapie obydwa dotknięcia zostały zakończone i adresy w pamięci mogą być ponownie wykorzystane przez system. To jest jedynie prosty przykład pokazujący jednoczesną obsługę dwóch dotknięć ekranu. Podczas przeprowadzania prób prawdopodobnie zobaczysz znacznie większą liczbę komunikatów wygenerowanych i wyświetlonych w oknie konsoli, a dotknięcia będą przechodziły przez wszystkie dostępne metody ich obsługi.

Poruszanie paletkami

Teraz przeprowadzimy modyfikację procedur obsługi dotknięć, aby zapewnić możliwość poruszania paletkami w poziomie wzdłuż osi X. W celu pobrania rzeczywistego położenia palca w widoku konieczne jest wywołanie metody `locationInView` obiektu `UITouch`. Wartością zwrotną wymienionej metody jest miejsce dotknięcia ekranu względem wskazanego widoku. Użyjemy widoku głównego, którego wymiary odpowiadają wymiarom ekranu. Wartość zwrotna jest typu `CGPoint`; to struktura składająca się z położenia X i Y. Ekran ma wysokość 480 pikseli, więc wartość Y punktu będzie użyta do określenia paletki, która powinna być poruszona. Jeżeli będzie dotyczyła górnej części ekranu (wartość mniejsza niż 240 pikseli), wówczas poruszona będzie paletka `paddle1`. W przeciwnym razie położenie zmieni paletka `paddle2`. Paletka powinna poruszać się jedynie wzdłuż osi X, więc zadaniem kodu jest ustalenie nowego centrum w punkcie określonym przez wartość X dotknięcia, podczas gdy wartość Y powinna pozostać bez zmian. Możesz wykorzystać `CGPointMake`, co jest szybkim sposobem inicjalizacji nowej struktury `CGPoint`. Wcześniejszą implementację metody `touchesBegan` zastąp przedstawionym poniżej kodem:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
   // Iteracja przez elementy dotknięcia.
   for (UITouch *touch in touches)
   {
      // Pobranie miejsca dotknięcia palcem w widoku.
      CGPoint touchPoint = [touch locationInView:self.view];

      // Poruszenie paletką na podstawie określenia połowy ekranu,
      // w której nastąpiło dotknięcie.
      if (touchPoint.y < 240)
      {
         viewPaddle1.center = CGPointMake(touchPoint.x, viewPaddle1.center.y);
      }
      else
      {
         viewPaddle2.center = CGPointMake(touchPoint.x, viewPaddle2.center.y);
      }
   }
}
```

Kod przedstawiony powyżej zajmuje się obsługą początku operacji dotknięcia, ale nie obsługuje sytuacji, w której palec użytkownika jest przesuwany po ekranie. Paletka jest obsługiwana poprzez położenie palca na ekranie, a następnie poruszanie nim w obydwie strony, co wymaga implementacji obsługi zdarzenia `touchesMoved`. W omawianym przypadku wystarczy po raz kolejny wywołać metodę `touchesBegan`, ponownie wykorzystując tę samą logikę obsługującą ruch paletki. Wcześniejszą implementację metody `touchesMoved` zastąp przedstawionym poniżej kodem:

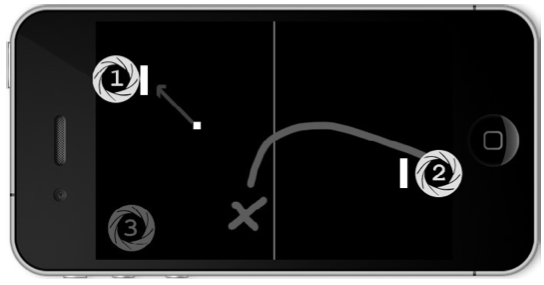
```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
   [self touchesBegan:touches withEvent:event];
}
```

Uruchom aplikację w urządzeniu i przekonaj się, że obiema paletkami możesz poruszać w tym samym czasie poprzez jednoczesne dotykanie ekranu w różnych miejscach. Nieźle jak na jedynie kilka wierszy kodu! Pobaw się przez chwilę paletkami i spróbuj znaleźć błędy w bieżącej implementacji. Implementacja zawiera dwa błędy, które zostaną poprawione w kolejnej sekcji.

Problemy z Multi-Touch: trzeci palec

Gracz kontrolujący paletkę najczęściej umieści palec pomiędzy paletką i krawędzią ekranu. Następnie będzie poruszał palcem w górę i w dół, nie podnosząc palca z ekranu aż do końca danej rundy gry. W bieżącej implementacji można dostrzec dwa problemy. Pierwszy to że gracz może przesunąć palec na połowę drugiego gracza i tym samym przejmie kontrolę nad jego paletką. Jak pokazano na rysunku 2.10, drugi gracz przesuwa palec na połowę pierwszego gracza, a następnie w lewą stronę. W ten sposób pierwszy gracz nie będzie miał pełnej kontroli nad poruszaniem swoją paletką, może przepuścić piłeczkę, a gracz drugi zdobędzie nieuczciwie punkt. Drugi problem polega na tym, że każde dodatkowe (poza pierwszymi dwoma) dotknięcie ekranu również wpływa na położenie paletki gracza. Trzeci palec na ekranie (również pokazany na rysunku 2.10) spowoduje, że pierwszy gracz utraci pełnię kontroli nad swoją paletką i drugi gracz może zyskać nieuczciwie

punkt. Obydwa wymienione problemy trzeba wyeliminować, aby gra nie cierpiała przez lata na skutek teorii spiskowych dotyczących tego, co tak naprawdę się zdarzyło podczas rozgrywki w aplikacji Paddles.



Rysunek 2.10. Problemy występujące w bieżącej implementacji gry

Pod względem sposobu kontroli paletek gra nie powinna się różnić od rzeczywistej. W rzeczywistej grze osoba bierze paletkę do ręki i trzyma ją aż do zakończenia gry. W budowanej przez nas grze oczekujemy takiego samego zachowania. Kiedy gracz zyskuje kontrolę nad paletką, wówczas drugi gracz nie powinien mieć możliwości zyskania kontroli nad tą paletką, chyba że pierwszy gracz mu na to pozwoli. Dlatego też paletka powinna ignorować wszelkie dodatkowe dotknięcia występujące na jej połowie ekranu. Poza tym jeśli gracz ma przypisaną paletkę, wtedy nie powinien uzyskać możliwości kontrolowania drugiej paletki po przeciągnięciu palca na połowę przeciwnika. W celu zaimplementowania takiej logiki trzeba wiedzieć, które dotknięcia dotyczą danej paletki. Jak już wcześniej wspomniano, obiekty `UITouch` zawsze będą miały te same egzemplarze w trakcie całego cyklu życiowego zdarzeń dotyku. Ten fakt można wykorzystać do powiązania określonego dotknięcia z daną paletką.

Właściwa obsługa dotknięć

Aby śledzić dotknięcia dotyczące danej paletki, konieczne jest dodanie kilku zmiennych do interfejsu `PaddlesViewController`. Zmienna `touch1` będzie aktywnym dotknięciem powiązanym z paletką `paddle1`, natomiast `touch2` — aktywnym dotknięciem powiązanym z paletką `paddle2`. Jeżeli paletka nie będzie przypisana do danego dotknięcia, wówczas wartością będzie `nil`. Przedstawione poniżej zmienne dodaj do definicji interfejsu *PaddlesViewController.h*:

```
@interface PaddlesViewController : UIViewController
{
   UITouch *touch1;
   UITouch *touch2;
}
```

Trzeba zmodyfikować implementację metody `touchesBegan` w celu przypisania paletki do określonego dotknięcia tylko wtedy, gdy pozostaje ono nieprzypisane. Nadal należy wykorzystywać logikę wymagającą, aby dotknięcie górnej części ekranu było przypisane do paletki `paddle1`, natomiast dolnej części ekranu — do paletki `paddle2`. Jeżeli wymienione warunki będą spełnione, nastąpi przypisanie odpowiedniego obiektu dotknięcia do paletki oraz przesunięcie paletki z zapisanego wcześniej położenia. Poprzednią implementację metody `touchesBegan` zastąp poniższym kodem:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
   // Iteracja przez elementy dotknięcia.
   for (UITouch *touch in touches)
   {
      // Pobranie miejsca dotknięcia palcem w widoku.
      CGPoint touchPoint = [touch locationInView:self.view];

      // Sprawdzenie, która połowa ekranu została dotknięta, i przypisanie
      // dotknięcia do określonej paletki, jeśli nie zostało jeszcze przypisane.
      if (touch1 == nil && touchPoint.y < 240)
      {
         touch1 = touch;
         viewPaddle1.center = CGPointMake(touchPoint.x, viewPaddle1.center.y);
      }
      else if (touch2 == nil && touchPoint.y >= 240)
      {
         touch2 = touch;
         viewPaddle2.center = CGPointMake(touchPoint.x, viewPaddle2.center.y);
      }
   }
}
```

Po przypisaniu każdej paletce odpowiednich dotknięć trzeba zapewnić obsługę poruszania paletkami. Nie możemy już dłużej wywoływać metody `touchesBegan`, ponieważ paletki posiadające przypisane dotknięcia zostaną zignorowane. Zamiast tego trzeba sprawdzić, czy jakikolwiek dostarczony obiekt dotknięcia odpowiada dowolnemu obiektowi dotknięcia przypisanego paletce. W przypadku uaktualnienia dotknięcia przypisanego paletce wiadomo, że można zmienić jej położenie. Wszelkie dotknięcia nieprzypisane paletce można bezpiecznie zignorować. Poprzednią implementację metody `touchesMoved` zastąp poniższym kodem:

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
   // Iteracja przez elementy dotknięcia.
   for (UITouch *touch in touches)
   {
      // Pobranie miejsca dotknięcia palcem w widoku.
      CGPoint touchPoint = [touch locationInView:self.view];

      // Jeżeli dotknięcie jest przypisane paletce, zmieniamy położenie paletki.
      if (touch == touch1)
      {
         viewPaddle1.center = CGPointMake(touchPoint.x, viewPaddle1.center.y);
      }
      else if (touch == touch2)
      {
         viewPaddle2.center = CGPointMake(touchPoint.x, viewPaddle2.center.y);
      }
   }
}
```

Trzeba jeszcze zapewnić obsługę sytuacji, w której użytkownik podniósł palec znad ekranu, czyli znad paletki z przypisanym dotknięciem. W tym celu należy zaimplementować metodę `touchesEnded`. Jeżeli dowolne z dotknięć w zbiorze odpowiada przypisanemu paletce, wówczas to przypisane powinno być usunięte poprzez ustawienie wartości `nil`.

Jeśli tego nie zrobisz, to prawdopodobnie utracisz kontrolę nad paletką po podniesieniu palca z ekranu. To odpowiada odłączeniu kontrolera od konsoli, co nie jest dobrym rozwiązaniem! Poprzednią implementację metody `touchesEnded` zastąp przedstawionym poniżej kodem:

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
   // Iteracja przez elementy dotknięcia.
   for (UITouch *touch in touches)
   {
      if (touch == touch1) touch1 = nil;
         else if (touch == touch2) touch2 = nil;
   }
}
```

Musisz się także upewnić o zapewnieniu obsługi przerwania zdarzenia; w tym przypadku można użyć kodu utworzonego dla metody `touchesEnded`. Pamiętaj, że ta metoda będzie wywołana po przerwaniu operacji dotknięcia. Tego rodzaju sytuację możesz przetestować, nawiązując połączenie z urządzeniem (iPhone) lub wywołując zdefiniowany alarm (iPod touch). Tuż przed wystąpieniem zakłócenia zdarzenia zauważysz, że wszystkie operacje dotknięcia zostały przerwane. Jeżeli nie zapewnisz obsługi metody przerwania zdarzenia, gracze prawdopodobnie nie będą mogli uzyskać kontroli nad paletkami po zakończeniu zdarzenia zakłócającego. Poprzednią implementację metody `touchesCancelled` zastąp przedstawionym poniżej kodem:

```
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
   [self touchesEnded:touches withEvent:event];
}
```

Uruchom aplikację w urządzeniu iOS i zwróć uwagę, że po przypisaniu dotknięcia paletce dodatkowe dotknięcie nie powoduje zmiany położenia paletki. Przekonaj się również, że przeciągnięcie palca na połowę przeciwnika nie powoduje uzyskania kontroli nad jego paletką. Gra ma więc solidną implementację obsługi wyświetlacza Multi-Touch.

Animacja

W tej chwili mamy paletki poruszające się w zależności od tego, który gracz dotyka ekranu. Nadeszła więc pora na wprawienie piłeczki w ruch. Gra wymaga pętli animacji poruszającej piłeczkę w określonym kierunku z daną prędkością. W interfejsie `PaddlesViewController` dodaj poniższe zmienne odpowiedzialne za śledzenie kierunku i prędkości poruszania się piłeczki:

```
float dx;
float dy;
float speed;
```

Zmienne `dx` i `dy` przedstawiają kierunek poruszania się piłeczki, natomiast zmienna `speed` jej prędkość. Prędkość i kierunek wolę śledzić za pomocą oddzielnych zmiennych, ponieważ to ułatwia zwiększanie prędkości piłeczki wraz z czasem trwania gry. Zmienna `dx` oznacza kierunek poruszania się piłeczki wzdłuż osi X. Wartość `-1` zmiennej `dx` oznacza ruch w lewą stronę, `0` oznacza brak ruchu, natomiast `1` to ruch w prawą stronę. Z kolei zmienna `dy` przedstawia poruszanie się piłeczki w pionie: wartość `-1` oznacza ruch w górę,

wartość 1 w dół. Kierunek może być określony także dowolną wartością z zakresu od -1 do 1, w takim przypadku piłeczka będzie poruszała się po ekranie pod dowolnym kątem.

Informacje o kierunku i prędkości poruszania się piłeczki trzeba zerować na początku każdej rundy, co wymaga dodania metody odpowiedzialnej za inicjalizację wspomnianych wartości. Na początku gry i każdej rundy zmienne określające położenie oraz kierunek poruszania się piłeczki powinny otrzymać losowo wygenerowane wartości. Ponownie wykorzystamy funkcję `arc4random()`, której użyliśmy w poprzednim rozdziale podczas budowy gry matematycznej. Jednak tym razem wybierana będzie wartość -1 lub 1. W pliku implementacji należy umieścić kod metody `reset`:

```
- (void)reset
{
    // Określenie kierunku poruszania się piłeczki: w lewo lub w prawo.
    if ((arc4random() % 2) == 0) dx = -1; else dx = 1;

    // Odwrócenie wartości dy, jeżeli nie wynosi 0, ponieważ to spowoduje ruch piłeczki
    // w kierunku gracza, który zdobył punkt. W przeciwnym razie kierunek będzie wybrany losowo.
    if (dy != 0) dy = -dy;
       else if ((arc4random() % 2) == 0) dy = -1; else dy = 1;

    // Przejście do losowo wybranego położenia na środku ekranu.
    viewPuck.center = CGPointMake(15 + arc4random() % (320-30), 240);

    // Wyzerowanie prędkości.
    speed = 2;
}
```

W powyższym kodzie zmienna `dx` otrzymuje losowo wybraną wartość: -1 lub 1. Tak więc na początku rundy piłeczka będzie poruszała się w lewo lub w prawo. Ponadto zmienna `dy` otrzyma wartość -1 lub 1, jeśli obecnie ma wartość 0. Jednak gdy jej wartość jest różna od zera, kierunek zostanie odwrócony. Przyjęto takie rozwiązanie, ponieważ na początku gry piłeczka powinna poruszać się w losowo wybranym kierunku. Jednak po zdobyciu punktu przez gracza 1 piłeczka powinna poruszać się w przeciwnym kierunku, aby gracz, który zdobył punkt, musiał uderzyć piłeczkę jako pierwszy.

Metoda zerująca zmienia także położenie piłeczki i umieszcza ją w dowolnym punkcie na linii środkowej. Prędkość początkowa wynosi 2; to liczba pikseli, o które piłeczka się przesuwa podczas pojedynczej klatki animacji. Dlaczego akurat 2? Początkowo wybrałem wartość 1, ale wówczas prędkość wydawała się zbyt mała.

Kolejnym krokiem jest dodanie prostej funkcji animacji odpowiedzialnej za przesunięcie piłeczki z jej położenia bieżącego do nowego z uwzględnieniem kierunku i szybkości. Funkcja animacji będzie nieustannie wywoływana podczas gry. Poniższy kod umieść pod metodą `reset`:

```
- (void)animate
{
    // Przesunięcie piłeczki do nowego położenia, uwzględniając jej kierunek i prędkość.
    viewPuck.center = CGPointMake(viewPuck.center.x + dx*speed,
       viewPuck.center.y + dy*speed);
}
```

Do interfejsu `PaddlesViewController` dodaj zmienną `NSTimer`:

```
NSTimer *timer;
```

Dodana zmienna będzie służyła do wywoływania funkcji animacji co $^{1}/_{60}$ sekundy, czyli z szybkością 60 klatek na sekundę. Kod gwarantuje także, że piłeczka będzie widoczna podczas rozpoczynania animacji i ukrywana po jej zatrzymaniu. W ten sposób uzyskujemy efekt „zabrania" piłeczki z pola gry, co jest użyteczne w przypadku wstrzymania gry. Odpowiednia logika zostanie dodana w dalszej części rozdziału.

W smartfonie iPhone ekran jest odświeżany z częstotliwością 60 Hz, czyli 60 razy na sekundę. Użycie szybszej animacji niż częstotliwość odświeżania ekranu może spowodować „przeskoczenie" pewnych klatek animacji. W celu uzyskania płynnej animacji najlepszym rozwiązaniem jest określenie jej szybkości na poziomie jak najbliższym częstotliwości odświeżania ekranu.

Przedstawiony poniżej kod umieść tuż za metodą `reset`. Dodany kod jest odpowiedzialny za uruchamianie i zatrzymywanie stopera animacji gry.

```
- (void)start
{
   if (timer == nil)
   {
      // Utworzenie stopera animacji.
      timer = [[NSTimer scheduledTimerWithTimeInterval:1.0/60.0
                                                target:self
                                                selector:@selector(animate)
                                                userInfo:NULL
                                                repeats:YES] retain];
   }

   // Wyświetlenie piłeczki.
   viewPuck.hidden = NO;
}

- (void)stop
{
   if (timer != nil)
   {
      [timer invalidate];
      [timer release];
      timer = nil;
   }

   // Ukrycie piłeczki.
   viewPuck.hidden = YES;
}
```

Kolejnym krokiem jest dodanie metody `viewDidLoad` poprzez usunięcie znaków komentarzy wokół kodu metody `viewDidLoad` znajdującego się już w pliku implementacji lub poprzez dodanie nowej na końcu pliku implementacji. W wymienionej metodzie będzie następowało wyzerowanie zmiennych gry i uruchomienie animacji. Metoda `viewDidLoad` to także odpowiednie miejsce do przeprowadzenia operacji inicjalizacyjnych, takich jak wczytanie widoku oraz połączenie właściwości widoku (paletki i piłeczka) w celu uzyskania możliwości dostępu do nich:

```
- (void)viewDidLoad
{
   [super viewDidLoad];
```

```
    [self reset];
    [self start];
}
```

Po uruchomieniu gry przekonasz się, że piłeczka jest szybko animowana poza ekran i nigdy nie powraca. W kolejnej sekcji dodasz więc kod odpowiedzialny za odbijanie piłeczki od ścian i paletek.

Wykrywanie kolizji

W grze konieczne jest wykrywanie kolizji piłeczki ze ścianami i paletkami. Dobrą wiadomością jest istnienie łatwego sposobu określenia, czy dwa widoki przecinają się ze sobą. Klasa `UIView` zawiera zmienną `frame` będącą strukturą `CGRect`, która przedstawia położenie i wielkość widoku. Dzięki użyciu funkcji `CGRectIntersectsRect` można określić, czy ramka przecina się z innymi widokami.

Konieczne jest utworzenie funkcji sprawdzającej, czy dany prostokąt przecina się z piłeczką. Jeżeli tak, kierunek piłeczki zostanie zmieniony. Nowy kierunek piłeczki będzie opcjonalny; podanie wartości `0` dla zmiennej `dx` lub `dy` nie spowoduje zmiany. Nad funkcją `animate` należy umieścić poniższy kod:

```
- (BOOL)checkPuckCollision:(CGRect)rect
                      DirX:(float)x
                      DirY:(float)y
{
    // Sprawdzenie, czy piłeczka przecina się z podanym prostokątem.
    if (CGRectIntersectsRect(viewPuck.frame, rect))
    {
        // Zmiana kierunku poruszania się piłeczki.
        if (x != 0) dx = x;
        if (y != 0) dy = y;
        return TRUE;
    }
    return FALSE;
}
```

W ten sposób utworzyliśmy elegancką, ogólną funkcję wykrywania kolizji. Pozostało już tylko dodanie implementacji wykrywania kolizji w metodzie `animate`. W przypadku lewej ściany należy utworzyć prostokąt obejmujący lewą stronę ekranu. Dlatego też zdefiniowanie prostokąta za pomocą polecenia `CGRectMake(-10,0,20,480)` powoduje utworzenie pionowego paska wzdłuż lewej krawędzi ekranu. Ściana tak naprawdę zaczyna się poza ekranem w punkcie –10 i ma grubość 20 pikseli, co oznacza, że połowa zdefiniowanego prostokąta znajduje się na ekranie. Podobną ścianę tworzymy po prawej stronie ekranu — `CGRectMake(310,0,20,480)`. Do zmiany kierunku piłeczki można użyć funkcji `fabs()` pobierającej wartość bezwzględną liczby zmiennoprzecinkowej. Jeżeli piłeczka uderzy w lewą ścianę, wówczas kierunek X zostania zmieniony na liczbę dodatnią, natomiast w przypadku uderzenia prawej ściany kierunek X będzie liczbą ujemną. W ten sposób uzyskujemy efekt odbicia piłeczki od ściany z taką samą prędkością. W obydwu przypadkach kierunek Y piłeczki jest ignorowany poprzez przekazanie wartości `0`. Nie chcesz, aby piłeczka została odbita w kierunku gracza, od którego przybyła. Dlatego też na końcu metody `animate` umieść przedstawiony poniżej kod:

```
// Sprawdzenie wystąpienia kolizji piłeczki z lewą lub prawą ścianą.
[self checkPuckCollision:CGRectMake(-10,0,20,480)
                    DirX:fabs(dx)
                    DirY:0];
[self checkPuckCollision:CGRectMake(310,0,20,480)
                    DirX:-fabs(dx)
                    DirY:0];
```

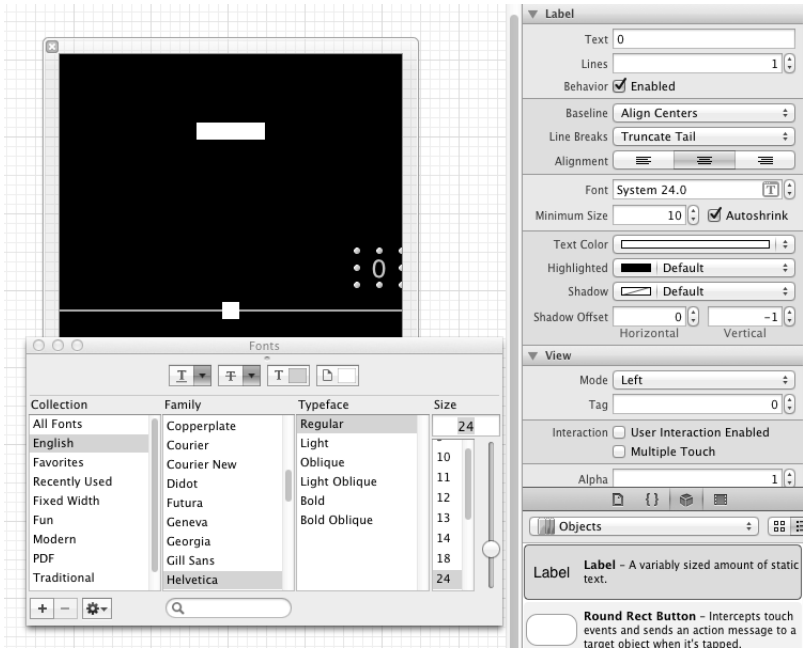
Do wykrywania kolizji z paletką można wykorzystać zmienną `frame` obiektów `viewPaddle1` i `viewPaddle2`. Kierunek Y piłeczki może być dostosowany do odbicia jej od paletki. Jeżeli piłeczka uderzy górną krawędź paletki, wówczas kierunek Y będzie zmieniony na `1`, co oznacza kierunek w dół ekranu. Natomiast uderzenie przez piłeczkę dolnej krawędzi paletki oznacza zmianę kierunku Y na `-1`, co oznacza kierunek w górę ekranu. Gra powinna dostosować także kierunek X piłeczki na podstawie miejsca uderzenia paletki gracza. Jeżeli piłeczka uderzy lewą stronę paletki, wówczas odbicie powinno nastąpić w lewą stronę. Natomiast uderzenie prawej strony paletki powinno spowodować odbicie piłeczki w prawą stronę. Po obliczeniu różnicy pomiędzy dwoma środkami obiektów wzdłuż osi X otrzyma się liczbę dodatnią lub ujemną — w zależności od miejsca uderzenia. Ponieważ paletka ma szerokość 64 pikseli, różnicę można podzielić przez 32 i znormalizować wartość wynikową pomiędzy `-1` i `1`. Przykładowo: jeśli środek piłeczki uderzy lewą stronę paletki, wówczas różnica pomiędzy obydwoma środkami wzdłuż osi X wyniesie –32. Po podzieleniu tej różnicy przez 32 wynikiem będzie wartość `-1` przypisana zmiennej `dx`, co przekłada się na odbicie piłeczki w lewą stronę. Poniższy kod umieść na końcu metody `animate`:

```
// Wykrywanie kolizji piłeczki z paletkami graczy.
[self checkPuckCollision:viewPaddle1.frame
                    DirX:(viewPuck.center.x - viewPaddle1.center.x) / 32.0
                    DirY:1];
[self checkPuckCollision:viewPaddle2.frame
                    DirX:(viewPuck.center.x - viewPaddle2.center.x) / 32.0
                    DirY:-1];
```

Po wprowadzeniu tych zmian uruchom grę i przekonaj się, że piłeczka odbija się od lewej i prawej ściany oraz od górnej i dolnej paletki. Zwróć także uwagę, że przejście piłeczki obok paletki powoduje opuszczenie przez nią pola gry, na które nigdy już nie wraca. Trzeba więc zająć się obsługą i tej sytuacji oraz zapewnić możliwość zdobywania punktów przez graczy.

Punktacja

W widoku trzeba umieścić kilka etykiet pozwalających na wyświetlenie punktacji. Przejdź do modułu Iterface Builder i ponownie otwórz plik *PaddlesViewController.xib*. Przeciągnij nową etykietę na widok, ustaw *0* jako jej tekst początkowy, wyśrodkuj go, a następnie zwiększ czcionkę do przynajmniej 24 punktów. Etykietę umieść przy prawej krawędzi ekranu. Następnie przejdź do inspektora wielkości i przenieś do centrum położenie punktu początkowego (*Origin*). Ustaw `200` jako wartość Y, co oznacza 40 pikseli nad linią środkową. Wynik powinien być podobny do pokazanego na rysunku 2.11.



Rysunek 2.11. Dodanie etykiet punktacji do widoku

Skopiuj i wklej etykietę oraz umieść ją poniżej linii środkowej, mniej więcej w takiej samej odległości. Upewnij się o dosunięciu etykiety do prawej krawędzi widoku, a następnie dla Y ustaw wartość 280, czyli 40 pikseli poniżej linii środkowej.

Podobnie jak wcześniej utwórz w kontrolerze widoku outlety dla nowych etykiet, co pozwoli Ci na uzyskiwanie do nich dostępu poprzez kod. Etykieta dla pierwszego gracza powinna nosić nazwę `viewScore1`, natomiast dla drugiego — `viewScore2`. Po wprowadzeniu zmian plik interfejsu będzie miał następującą postać:

```
@interface PaddlesViewController : UIViewController
{
   UITouch *touch1;
   UITouch *touch2;

   float dx;
   float dy;
   float speed;
   NSTimer *timer;
}

@property (nonatomic, retain) IBOutlet UIView *viewPaddle1;
@property (nonatomic, retain) IBOutlet UIView *viewPaddle2;
@property (nonatomic, retain) IBOutlet UIView *viewPuck;
@property (nonatomic, retain) IBOutlet UILabel *viewScore1;
@property (nonatomic, retain) IBOutlet UILabel *viewScore2;

@end
```

Konieczne jest dodanie funkcji sprawdzającej, czy którykolwiek z graczy wywalczył punkt. Jeżeli piłeczka uderzy w górną krawędź ekranu, punkt wywalczył gracz numer dwa.

Natomiast jeśli piłeczka uderzy w dolną krawędź ekranu, punkt zyskuje gracz numer jeden. W celu dodania punktu wartości obydwu etykiet punktacji muszą zostać skonwertowane na liczby całkowite. Klasa `NSString` posiada metodę o nazwie `intValue`, której wartością zwrotną jest liczba całkowita z podanego ciągu tekstowego.

> Metoda `intValue` zawsze zwraca liczbę całkowitą, niezależnie od zawartości ciągu tekstowego. Jeżeli ciąg tekstowy zawiera inne dane niż liczba, na przykład litery bądź symbole, wartością zwrotną będzie 0. Jeśli na początku ciągu tekstowego znajdzie się puste miejsce, zostanie pominięte. Metoda ponadto zwraca wartość `INT_MAX` lub `INT_MIN` w przypadku przepełnienia.

Wartości liczbowe obydwu etykiet będą przechowywane w zmiennych `s1` i `s2`, a następnie zwiększane o jeden po wywalczeniu punktu przez gracza. Ostatnim krokiem jest konwersja wartości liczbowej z powrotem na ciąg tekstowy, uaktualnienie wartości etykiet punktacji i wyzerowanie rundy. Funkcja `checkGoal` określa także, czy punkt został zaliczony, czy nie. Poniższy kod umieść przed funkcją `animate`:

```
- (BOOL)checkGoal
{
    // Sprawdzenie, czy piłeczka wykroczyła poza ekran. Jeśli tak, zerujemy rundę.
    if (viewPuck.center.y < 0 || viewPuck.center.y >= 480)
    {
        // Pobranie wartości liczbowej z etykiet punktacji.
        int s1 = [viewScore1.text intValue];
        int s2 = [viewScore2.text intValue];

        // Przydzielenie punktu odpowiedniemu graczowi.
        if (viewPuck.center.y < 0) ++s2; else ++s1;

        // Uaktualnienie etykiet punktacji.
        viewScore1.text = [NSString stringWithFormat:@"%u", s1];
        viewScore2.text = [NSString stringWithFormat:@"%u", s2];

        // Wyzerowanie rundy.
        [self reset];

        // Zwrócenie wartości TRUE w przypadku zdobycia punktu.
        return TRUE;
    }

    // Brak punktu.
    return FALSE;
}
```

Następnie wywołanie funkcji `checkGoal` trzeba umieścić na końcu metody `animate`:

```
- (void)animate
{
    // Przesunięcie piłeczki do nowego położenia, uwzględniając jej kierunek i prędkość.
    viewPuck.center = CGPointMake(viewPuck.center.x + dx*speed,
                                  viewPuck.center.y + dy*speed);

    // Sprawdzenie wystąpienia kolizji piłeczki z lewą lub prawą ścianą.
    [self checkPuckCollision:CGRectMake(-10,0,20,480)
                        DirX:fabs(dx)
                        DirY:0];
```

```
    [self checkPuckCollision:CGRectMake(310,0,20,480)
                      DirX:-fabs(dx)
                      DirY:0];

    // Wykrywanie kolizji piłeczki z paletkami graczy.
    [self checkPuckCollision:viewPaddle1.frame
                      DirX:(viewPuck.center.x - viewPaddle1.center.x) / 32.0
                      DirY:1];
    [self checkPuckCollision:viewPaddle2.frame
                      DirX:(viewPuck.center.x - viewPaddle2.center.x) / 32.0
                      DirY:-1];

    // Sprawdzenie, czy został zdobyty punkt.
    [self checkGoal];
}
```

Uruchom grę i podziwiaj przyznawanie punktów oraz zerowanie rundy po przejściu piłeczki przez górną lub dolną krawędź ekranu. W ten sposób z powodzeniem zaimplementowaliśmy działający system punktacji, ale obecnie gra nigdy się nie kończy. Kolejnym krokiem będzie dodanie warunku kończącego grę oraz ostateczne jej wykończenie.

Wykończenie gry

Grę trzeba jeszcze dopracować, między innymi dodać komunikat wyświetlany po wygraniu rundy przez gracza, dać graczom możliwość przygotowania się przed rozpoczęciem gry, zwiększać trudność gry po każdym uderzeniu piłeczki oraz dodać możliwość wstrzymywania i wznawiania gry.

Wyświetlanie komunikatów

Najłatwiejszym sposobem wyświetlenia użytkownikowi komunikatu jest wykorzystanie okna komunikatu. Wspomniane okno powoduje wyświetlenie komunikatu i wymaga naciśnięcia przycisku zamykającego okno. Używana do wyświetlenia okna klasa `UIAlertView` ma bardzo prosty interfejs.

W pliku interfejsu umieść poniższy wiersz kodu:

```
UIAlertView *alert;
```

Teraz należy utworzyć funkcję pobierającą treść komunikatu i wyświetlającą go użytkownikowi. Funkcja powinna również zatrzymać animację, co efektywnie spowoduje wstrzymanie gry na czas wyświetlania komunikatu. Oprócz tego przed wyświetleniem komunikatu funkcja sprawdzi, czy w danej chwili jest wyświetlany jakikolwiek inny komunikat. Przedstawiony poniżej kod umieść za funkcją `stop`:

```
- (void)displayMessage:(NSString*) msg
{
    // Jednocześnie może być wyświetlony tylko jeden komunikat.
    if (alert) return;

    // Wstrzymanie animacji.
    [self stop];

    // Utworzenie i wyświetlenie okna komunikatu.
```

```
    alert = [[UIAlertView alloc] initWithTitle:@"Gra"
                                       message:msg
                                      delegate:self
                             cancelButtonTitle:@"OK"
                             otherButtonTitles:nil];
    [alert show];
    [alert release];
}
```

Powyższą funkcję możesz wykorzystać po uruchomieniu gry w celu umożliwienia graczom przygotowania się do rozgrywki. Utworzymy także funkcję o nazwie `newGame` odpowiedzialną za wyzerowanie rundy, ustawienie wartości 0 punktacji oraz wyświetlenie komunikatu o treści *Gotowi do gry?*. Poniżej funkcji `displayMessage` w pliku implementacji umieść kod wymienionej funkcji `newGame`:

```
- (void)newGame
{
    [self reset];

    // Wyzerowanie punktacji.
    viewScore1.text = [NSString stringWithString:@"0"];
    viewScore2.text = [NSString stringWithString:@"0"];

    // Wyświetlenie komunikatu pozwalającego na rozpoczęcie gry.
    [self displayMessage:@"Gotowi do gry?"];
}
```

Metoda `viewDidLoad` do doskonałe miejsce na wyświetlenie komunikatu pozwalającego graczom na rozpoczęcie rozgrywki. Można w tym miejscu usunąć stoper animacji, ponieważ gra oficjalnie się rozpocznie po naciśnięciu przycisku *OK*. Istniejący kod w metodzie `viewDidLoad` zastąp poniższym:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self newGame];
}
```

Po naciśnięciu przycisku przez użytkownika klasa `UIAlertView` wywoła delegata. Jeśli komunikat zawiera więcej niż tylko jeden przycisk, wówczas trzeba sprawdzić wartość parametru `buttonIndex` w celu ustalenia, który przycisk został naciśnięty. W omawianym przypadku mamy tylko pojedynczy przycisk *OK*, więc nie ma potrzeby sprawdzania. Aby zapewnić obsługę wywołania zwrotnego, tuż za definicją funkcji `newGame` umieść następujący kod:

```
- (void)alertView:(UIAlertView *)alertView
    didDismissWithButtonIndex:(NSInteger)buttonIndex
{
    // Okno zostało zamknięte, więc można rozpocząć grę.
    alert = nil;

    // Wyzerowanie rundy.
    [self reset];

    // Rozpoczęcie animacji.
    [self start];
}
```

Powyższy kod spowoduje wyzerowanie wartości zmiennej `alert`, wyzerowanie zmiennych rundy oraz rozpoczęcie odtwarzania animacji. Po uruchomieniu gry zostanie wyświetlony komunikat pozwalający graczom na przygotowanie się do rozgrywki przed faktycznym jej rozpoczęciem. Tę samą logikę wykorzystamy do wyświetlenia komunikatu *Koniec gry*.

Koniec gry

Trzeba określić liczbę punktów, której osiągnięcie przez dowolnego gracza spowoduje zakończenie gry i ogłoszenie zwycięzcy. Zalecam ustawienie tej wartości na minimum, aby można było łatwo przetestować warunek kończący grę. W takich przypadkach najczęściej korzystam z dyrektywy `#define`, która umożliwia łatwe zwiększenie wartości w przyszłości. Wspomnianą wartość prawdopodobnie trzeba będzie zwiększyć w przyszłości lub nawet umożliwić graczom jej ustalenie. Na początku pliku implementacji umieść poniższy wiersz kodu:

```
#define MAX_SCORE 3
```

Kolejnym krokiem jest dodanie poniższej metody, w której następuje sprawdzenie, czy gra powinna się oficjalnie zakończyć. Wartości obydwu etykiet punktacji są konwertowane na liczby, a następnie kod sprawdza, czy którykolwiek z graczy zdobył zdefiniowaną maksymalną możliwą do zdobycia liczbę punktów. Metoda informuje także, który gracz zdobył tę liczbę punktów. W pliku implementacji umieść przedstawiony poniżej kod:

```
- (int)gameOver
{
   if ([viewScore1.text intValue] >= MAX_SCORE) return 1;
   if ([viewScore2.text intValue] >= MAX_SCORE) return 2;
   return 0;
}
```

Następnie musimy zmodyfikować metodę wyświetlającą okno komunikatu w celu dodania operacji sprawdzenia, czy gra została zakończona. Jeżeli gra została zakończona, wówczas rozpoczynamy nową i pytamy graczy, czy są gotowi do ponownego podjęcia rozgrywki:

```
- (void)alertView:(UIAlertView *)alertView
   didDismissWithButtonIndex:(NSInteger)buttonIndex
{
   // Okno zostało zamknięte, więc można wyzerować grę i rozpocząć odtwarzanie animacji.
   alert = nil;

   // Sprawdzenie, czy należy rozpocząć nową grę.
   if ([self gameOver])
   {
      [self newGame];
      return;
   }

   // Wyzerowanie rundy.
   [self reset];

   // Wyzerowanie animacji.
   [self start];
}
```

Trzeba również zmodyfikować funkcję odpowiedzialną za przyznawanie punktu graczowi i dodać logikę sprawdzającą, czy gracz wygrał grę. Jeżeli zdefiniowana liczba punktów wymaganych do wygrania gry nie została jeszcze zdobyta, wówczas podobnie jak wcześniej jedynie zerujemy rundę. W metodzie `checkGoal` dodaj wiersze, które poniżej zostały pogrubione:

```
- (BOOL)checkGoal
{
    // Sprawdzenie, czy piłeczka wykroczyła poza ekran. Jeśli tak, zerujemy rundę.
    if (viewPuck.center.y < 0 || viewPuck.center.y >= 480)
    {
        // Pobranie wartości liczbowej z etykiet punktacji.
        int s1 = [viewScore1.text intValue];
        int s2 = [viewScore2.text intValue];

        // Przydzielenie punktu odpowiedniemu graczowi.
        if (viewPuck.center.y < 0) ++s2; else ++s1;

        // Uaktualnienie etykiet punktacji.
        viewScore1.text = [NSString stringWithFormat:@"%u", s1];
        viewScore2.text = [NSString stringWithFormat:@"%u", s2];

        // Sprawdzenie zwycięzcy.
        if ([self gameOver] == 1)
        {
            // Ogłoszenie zwycięzcy.
            [self displayMessage:@"Wygrał gracz numer 1!"];
        }
        else if ([self gameOver] == 2)
        {
            // Ogłoszenie zwycięzcy.
            [self displayMessage:@"Wygrał gracz numer 2!"];
        }
        else
        {
            // Wyzerowanie rundy.
            [self reset];
        }

        // Zwrócenie wartości TRUE w przypadku zdobycia punktu.
        return TRUE;
    }

    // Brak punktu.
    return FALSE;
}
```

Uruchom grę i prowadź rozgrywkę do chwili, gdy jeden z graczy zdobędzie trzy punkty. Na tym etapie gra powinna ogłosić zwycięzcę. Po zamknięciu okna gra zostanie wyzerowana i gracze będą mogli rozpocząć rozgrywkę od początku.

Zwiększanie poziomu trudności

Bardzo ważne jest znalezienie właściwego sposobu na zwiększanie poziomu trudności gry w trakcie rozgrywki. Na obecnym etapie gra jest bardzo łatwa i istnieje możliwość,

że żaden z graczy nigdy nie „przepuści" piłeczki. Poniżej znajduje się lista, którą przygotowałem, zastanawiając się nad sposobem zwiększenia poziomu trudności gry:

- zwiększenie prędkości piłeczki;
- zmniejszenie wielkości piłeczki;
- zmniejszenie szerokości paletek;
- umieszczenie obiektów dodatkowych na trasie poruszania się piłeczki.

Obecnie w grze mamy zmienną określającą szybkość poruszania się piłeczki i jest ona zerowana na początku każdej rundy. Zmienna określająca prędkość jest stosowana również podczas ustalania kierunku poruszania się piłeczki podczas jej animacji. Zwiększenie prędkości poruszania się piłeczki po każdym uderzeniu jej paletką wydaje się proste do implementacji. W tym miejscu trzeba pamiętać o jednej ważnej kwestii: istnieje prędkość maksymalna, z jaką może poruszać się piłeczka — jej przekroczenie może spowodować „przeskakiwanie" piłeczki ponad paletkami lub prostokątami użytymi w charakterze ścian po lewej i prawej stronie ekranu. Paletki mają wysokość 16 pikseli, natomiast szerokość ścian wynosi 20 pikseli. Jeżeli szybkość piłeczki będzie większa od podanych wartości, wówczas istnieje niebezpieczeństwo przeskoczenia tych obiektów, a tym samym nie dojdzie do wykrycia kolizji. Dlatego też trzeba ograniczyć prędkość poruszania się piłeczki do maksymalnie 10 pikseli na ramkę, co gwarantuje zapobiegnięcie występowaniu wspomnianej sytuacji przeskakiwania obiektów. Przed metodą `animate` dodaj przedstawiony poniżej kod:

```
- (void)increaseSpeed
{
   speed += 0.5;
   if (speed > 10) speed = 10;
}
```

Pamiętasz funkcję obsługującą wykrywanie kolizji i zwracającą wartość `TRUE` po wykryciu kolizji? Umieszczenie w niej dodatkowej logiki było celowym zabiegiem i pozwala na zapewnienie obsługi jeszcze innym operacjom. Zmodyfikuj metodę `animate`, aby prędkość poruszania się piłeczki była zwiększana po każdym wykryciu kolizji z paletką:

```
// Wykrywanie kolizji piłeczki z paletkami graczy.
if ([self checkPuckCollision:viewPaddle1.frame
                    DirX:(viewPuck.center.x - viewPaddle1.center.x) / 32.0
                    DirY:1])
{
   [self increaseSpeed];
}
if ([self checkPuckCollision:viewPaddle2.frame
                    DirX:(viewPuck.center.x - viewPaddle2.center.x) / 32.0
                    DirY:-1])
{
   [self increaseSpeed];
}
```

Uruchom grę i zwróć uwagę, że prędkość poruszania się piłeczki wzrasta po każdym uderzeniu paletką. Zobacz, jak długo możesz grać, zanim poziom trudności zwiększy się na tyle, że gracz zacznie popełniać błędy. Jeżeli uważasz, że gra nie jest dość trudna, spróbuj zwiększyć maksymalną dozwoloną prędkość piłeczki lub dodaj kolejne elementy z przedstawionej wcześniej listy (na przykład skrócenie paletek), które jeszcze bardziej utrudnią rozgrywkę.

Wstrzymywanie i wznawianie gry

Gry z reguły pozwalają graczom na wstrzymanie ich oraz wznowienie. Istnieje kilka sposobów obsługi wymienionej funkcjonalności:

- naciśnięcie przycisku zablokowania ekranu;
- nadejście połączenia przychodzącego, wiadomości tekstowej lub uruchomienie alarmu;
- naciśnięcie przycisku *Początek* i uruchomienie innej aplikacji.

Na początek trzeba do kontrolera widoku dodać kilka metod publicznych, które pozwolą na wstrzymywanie i wznawianie gry. Wstrzymanie gry spowoduje po prostu zatrzymanie stopera animacji, natomiast jej wznowienie spowoduje wyświetlenie komunikatu z pytaniem o wznowienie wstrzymanej gry. Kiedy gracz naciśnie przycisk *OK* w oknie komunikatu, gra spowoduje ponowne uruchomienie rundy.

W interfejsie kontrolera, po definicjach właściwości, ale jeszcze przed dyrektywą `@end` widoku, umieść poniższe deklaracje funkcji:

```
- (void)resume;
- (void)pause;
```

Implementacja metody `pause` polega po prostu na zatrzymaniu stopera animacji. Z kolei implementacja metody `resume` powoduje wyświetlenie komunikatu graczom. Na końcu pliku implementacji umieść poniższy kod:

```
- (void)pause
{
   [self stop];
}

- (void)resume
{
   // Wyświetlenie komunikatu z pytaniem o wznowienie gry.
   [self displayMessage:@"Gra jest wstrzymana"];
}
```

Delegat aplikacji posiada dwie metody wywołania zwrotnego, które informują, kiedy aplikacja staje się aktywna i nieaktywna. To zatem najlepsze miejsce do wywołania nowo dodanych metod `pause` i `resume`. W ten sposób zostaną obsłużone wszystkie wymienione wcześniej zdarzenia, które powinny spowodować wstrzymanie gry: zablokowanie ekranu, przerwanie spowodowane przez system lub naciśnięcie przycisku *Początek*.

> Aplikacje w urządzeniach działających pod kontrolą systemu iOS 4 i nowszego po naciśnięciu przycisku *Początek* przechodzą do stanu zatrzymania. Po wznowieniu aplikacji nadal będzie ona posiadała wszystkie istniejące stany gry, co oznacza, że logika metod wstrzymywania i wznawiania będzie działała. Natomiast w przypadku urządzeń działających pod kontrolą systemu iOS 3 i starszego naciśnięcie przycisku *Początek* powoduje zakończenie działania aplikacji. Jeżeli chcesz zapewnić obsługę metod wstrzymywania i wznawiania w tych urządzeniach, musisz przechowywać stan gry, a następnie wczytać go po ponownym uruchomieniu aplikacji.

Wywołania metod `pause` i `resume` umieść w metodach `applicationDidBecomeActive` i `applicationWillResignActive` pliku implementacji *PaddlesAppDelegate.m*:

```
- (void)applicationWillResignActive:(UIApplication *)application
{
```

```
    [self.viewController pause];
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    [self.viewController resume];
}
```

Uruchom grę i przetestuj różne możliwe sytuacje, na przykład zablokowanie i odblokowanie ekranu. Zablokowanie ekranu można zasymulować także w symulatorze urządzenia iOS poprzez naciśnięcie klawiszy *Command+L* lub wybranie opcji menu *Hardware/Lock*. Obsługiwane jest również szybkie przełączanie aplikacji, więc możesz nacisnąć przycisk *Początek*, a następnie ponownie uruchomić aplikację. Jedynym sposobem właściwego przetestowania sytuacji nadejścia rozmowy przychodzącej, wiadomości tekstowej lub uaktywnienia alarmu jest użycie rzeczywistego urządzenia iOS obsługującego wymienione funkcje.

Gest wstrząśnięcia

Teraz zajmiemy się innym sposobem wprowadzania danych w urządzeniach iOS: ruchem. Poznałeś już sposoby obsługi dotknięć, zdarzenia ruchu są równie łatwe w obsłudze. Kiedy urządzenie zmienia swoje położenie, mechanizm sprzętowy informuje o liniowej zmianie przyśpieszenia względem osi podstawowej w przestrzeni trójwymiarowej. Możesz więc otrzymywać ciągłe dane ruchu jako serię wartości `x`, `y`, `z`, ale to wymaga analizy każdego otrzymanego punktu danych i utworzenia algorytmu oceniającego, czy doszło do gestu wstrząśnięcia. Jeżeli każdy programista musiałby samodzielnie zaimplementować algorytm wykrywania wstrząsu, wówczas każda aplikacja wykorzystująca ten gest prawdopodobnie byłaby zaimplementowana inaczej, a skutkiem byłoby zakłopotanie użytkowników.

W systemie iOS 3.0 firma Apple zdecydowała się na ułatwienie zadania programistom i zaimplementowała koncepcję zdarzeń ruchu. Zdarzenia ruchu wykorzystują urządzenie przyśpieszeniomierza lub żyroskopu w celu obliczania rodzaju ruchu, który jest wykonywany przez urządzenie. W chwili powstawania tej książki obsługiwane było tylko jedno zdarzenie ruchu: wstrząśnięcie. Użycie zdarzenia wstrząśnięcia — w przeciwieństwie do implementacji własnego — pozwala użytkownikowi końcowemu na korzystanie ze spójnego gestu w różnych aplikacjach.

Podczas wstrząsania urządzeniem system przetwarza dane przyśpieszeniomierza i ustala, czy doszło do użycia gestu wstrząśnięcia. System informuje o chwilach rozpoczęcia i zakończenia zdarzenia ruchu. Nie informuje o każdym poszczególnym ruchu, a jedynie kiedy ogólnie zaczyna się i kończy ruch. Przykładowo: jeśli urządzeniem szybko wstrząśniesz trzykrotnie, wówczas możesz otrzymać informację o wykryciu pojedynczego ruchu wstrząsu.

Aby zaimplementować gest wstrząśnięcia, przede wszystkim kontroler widoku powinien otrzymać status *First Responder*. To może wydawać Ci się znajome, ponieważ wymieniony status zastosowaliśmy już w grze matematycznej utworzonej w rozdziale 1. W tamtej aplikacji kontrolka otrzymywała status *First Responder* w celu wyświetlenia klawiatury bez konieczności wcześniejszego naciśnięcia pola tekstowego przez użytkownika. Aby użyć gestu wstrząśnięcia, trzeba zastosować to samo rozwiązanie, choć tym razem status *First Responder* otrzyma kontroler widoku, a nie pojedyncza kontrolka. Trzeba również dodać metodę informującą system o otrzymaniu statusu *First Responder* przez kontrolera widoku.

Najlepsza chwila na nadanie statusu *First Responder* jest w momencie wyświetlania widoku. Podczas usuwania widoku z ekranu kontroler widoku powinien utracić status *First Responder*. W pliku *PaddlesViewController.m* umieść przedstawiony poniżej kod:

```
- (BOOL)canBecomeFirstResponder
{
   return YES;
}
```

Następnie w poniższy sposób zmodyfikuj metody `viewDidAppear` i `viewWillDisappear`:

```
-(void)viewDidAppear:(BOOL)animated
{
   [super viewDidAppear:animated];
   [self becomeFirstResponder];
}

- (void)viewWillDisappear:(BOOL)animated
{
   [self resignFirstResponder];
   [super viewWillDisappear:animated];
}
```

W ten sposób kontroler widoku został przygotowany do obsługi zdarzeń ruchu. Do obsługi zdarzeń ruchu służą trzy metody o nazwach `motionBegan`, `motionEnded` oraz `motionCancelled`. Są one podobne do metod zapewniających obsługę dotyku pod tym względem, że informują o rozpoczęciu oraz zakończeniu bądź przerwaniu zdarzenia ruchu. Zwróć uwagę na brak metody informującej o zmianie zdarzenia ruchu — obsługiwane jest pełne zdarzenie ruchu i nic innego. Umieść w pliku poniższy kod metod, który pozwoli Ci na poznanie sposobu działania omawianych metod poprzez wyświetlanie przez nie komunikatów w oknie konsoli:

```
- (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
   if (event.type == UIEventSubtypeMotionShake)
   {
      NSLog(@"Rozpoczęcie gestu wstrząśnięcia");
   }
}

- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
   if (event.type == UIEventSubtypeMotionShake)
   {
      NSLog(@"Zakończenie gestu wstrząśnięcia ");
   }
}

- (void)motionCancelled:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
   if (event.type == UIEventSubtypeMotionShake)
   {
      NSLog(@"Przerwanie gestu wstrząśnięcia ");
   }
}
```

Uruchom aplikację w symulatorze iOS, a następnie wybierz opcję menu *Hardware/Shake Gesture*. W konsoli modułu usuwania błędów pojawią się komunikaty podobne do poniższych:

```
2011-05-21 16:14:22.196 Paddles[28765:207] Rozpoczęcie gestu wstrząśnięcia
2011-05-21 16:14:22.198 Paddles[28765:207] Zakończenie gestu wstrząśnięcia
```

Zwróć uwagę, że w przypadku symulatora następuje niemal natychmiastowe wywołanie metod `motionBegan` i `motionEnd`. W symulatorze nie ma możliwości przetestowania sytuacji przerwania gestu wstrząśnięcia, więc jeśli chcesz przetestować i to zdarzenie, wówczas aplikację musisz uruchomić w rzeczywistym urządzeniu iOS. Możesz zauważyć, że komunikat *Zakończenie gestu wstrząśnięcia* pojawi się znacznie później niż *Rozpoczęcie gestu wstrząśnięcia*. Czasami można zobaczyć także komunikat *Przerwanie gestu wstrząśnięcia*, zwłaszcza po rozpoczęciu wstrząśnięcia urządzeniem w jednym kierunku, a następnie wstrzymaniu tej operacji na kilka sekund:

```
2011-05-21 16:25:34.669 Paddles[7830:707] Rozpoczęcie gestu wstrząśnięcia
2011-05-21 16:25:35.273 Paddles[7830:707] Zakończenie gestu wstrząśnięcia
2011-05-21 16:25:36.074 Paddles[7830:707] Rozpoczęcie gestu wstrząśnięcia
2011-05-21 16:25:38.547 Paddles[7830:707] Przerwanie gestu wstrząśnięcia
```

Na potrzeby budowanej tutaj gry można bezpiecznie przyjąć założenie, że rozpoczęcie ruchu oznacza przerwanie gry w jakikolwiek sposób. Być może jeden z graczy przypadkowo trącił iPhone'a lub ktoś szybko chwycił smartfona i umieścił go w kieszeni. W takich sytuacjach najlepszym rozwiązaniem będzie wstrzymanie gry. Metody dodane przed chwilą na potrzeby testów umieść w komentarzu lub całkowicie zastąp je poniższym kodem:

```
- (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
   if (event.type == UIEventSubtypeMotionShake)
   {
      // Wstrzymanie gry, a następnie jej wznowienie w celu wyświetlenia komunikatu.
      [self pause];
      [self resume];
   }
}
```

Jak widzisz, w tej samej metodzie są wywoływane metody `pause` i `resume`. Początkowo utworzyłem oddzielne metody do wstrzymywania i wznawiania gry, co wynikało z możliwych przerwań gry. Zwykle jest jedna metoda rozpoczynająca przerwanie oraz kolejna wznawiająca działanie po zakończeniu przerwania. Jednak w tym konkretnym przypadku wszystko jest przeprowadzane w pojedynczym kroku i dlatego też trzeba wywołać obydwie metody.

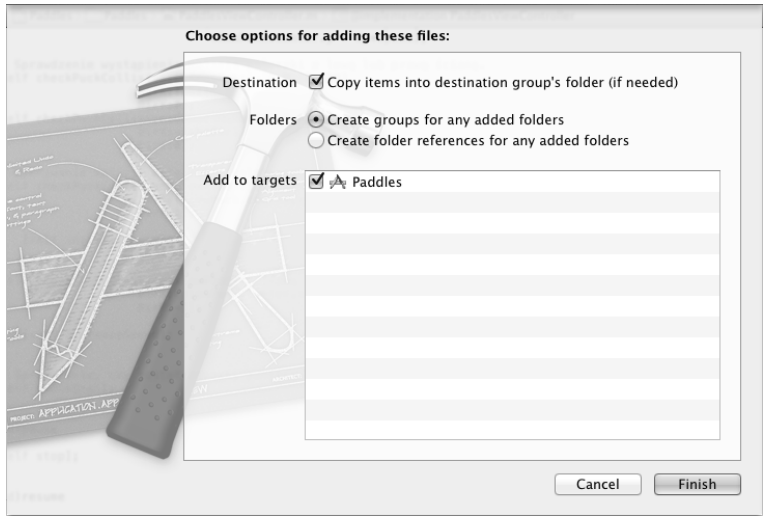
Uruchom grę w urządzeniu i upewnij się o wstrzymywaniu działania gry po wstrząśnięciu urządzeniem. Przedstawiony kod możesz przejrzeć ponownie później po dodaniu gracza sterowanego przez komputer i zmodyfikować go do zyskania dodatkowej mocy po wstrząśnięciu urządzeniem. Na razie wstrząśnięcie urządzeniem będzie powodowało wstrzymanie gry.

Dźwięk

Odtworzenie prostych efektów dźwiękowych jest możliwe za pomocą mechanizmu System Audio Services. Zaleca się, aby dźwięki były krótkie. Poniżej przedstawiono pozostałe wskazówki dotyczące dźwięków w aplikacji iOS:

- muszą być plikami w formacie *.caf*, *.aif* lub *.wav*;
- dane audio w pliku muszą być w formacie PCM lub IMA/ADPCM (IMA4);
- dźwięk zapisany w pliku audio może trwać maksymalnie 30 sekund.

W grze wykorzystamy trzy różne dźwięki odtwarzane podczas uderzenia piłeczki o ścianę, uderzenia piłeczki o paletkę oraz po zdobyciu punktu przez gracza. Przygotowałem wcześniej kilka plików dźwiękowych gotowych do wykorzystania w grze; możesz je pobrać z witryny O'Reilly *http://shop.oreilly.com/product/0636920018414.do* bądź mojej *http://toddmoore.com/*. Przygotowane pliki noszą nazwy *wall.wav*, *paddle.wav* oraz *score.wav*. Po pobraniu archiwum ZIP należy je rozpakować, a pliki przeciągnąć do projektu Xcode. Po wyświetleniu okna dialogowego z pytaniem upewnij się o zaznaczeniu pola wyboru *Copy items into destination group's folder (if needed)*, jak pokazano na rysunku 2.12, i kliknij przycisk *Finish*. W ten sposób pliki dźwiękowe zostaną dołączone do aplikacji i będziesz miał do nich dostęp z poziomu kodu.



Rysunek 2.12. Kopiowanie zasobów do projektu

Konieczne jest również dodanie struktury AudioToolbox do aplikacji. Kliknij plik projektu, a następnie *Paddles* w sekcji *Target*. Przejdź na kartę *Build Phases* i rozwiń *Link Binary With Libraries*. Po kliknięciu ikony plusa znajdującej się w dolnej części sekcji (zobacz rysunek 2.13) zobaczysz okno dialogowe, w którym będziesz mógł wybrać strukturę AudioToolbox. Od teraz aplikacja może korzystać z dodanej biblioteki.

Kolejnym krokiem jest dodanie pliku nagłówkowego *AudioToolbox.h* na początku pliku nagłówkowego *PaddlesViewController.h*:

```
#import <UIKit/UIKit.h>
#import "AudioToolbox/AudioToolbox.h"
```

Niezbędne jest także utworzenie tablicy przeznaczonej do przechowywania identyfikatorów dźwięków, które będą nadane po wczytaniu każdego pliku dźwiękowego. W interfejsie *PaddlesViewController.h* umieść więc następujący wiersz kodu:

```
SystemSoundID sounds[3];
```

Rysunek 2.13. Dodawanie struktury AudioToolbox

Po dodaniu tablicy do pliku nagłówkowego konieczne jest zaimplementowanie funkcji odpowiedzialnej za wczytanie plików dźwiękowych i przechowywanie wyniku w tablicy. Na początku pliku implementacji dodaj poniższy kod:

```
#define SOUND_WALL   0
#define SOUND_PADDLE 1
#define SOUND_SCORE  2

// Wczytanie efektu dźwiękowego do tablicy dźwięków.
- (void)loadSound:(NSString*)name Slot:(int)slot
{
   if (sounds[slot] != 0) return;

   // Utworzenie ścieżki dostępu do pliku dźwięku.
   NSString *sndPath = [[NSBundle mainBundle] pathForResource:name
                                                       ofType:@"wav"
                                                  inDirectory:@"/"];

   // Utworzenie identyfikatora dla dźwięku znajdującego się w slocie.
   AudioServicesCreateSystemSoundID((CFURLRef)
      [NSURL fileURLWithPath: sndPath], &sounds[slot]);
}

- (void)initSounds
{
   [self loadSound:@"wall" Slot:SOUND_WALL];
   [self loadSound:@"paddle" Slot:SOUND_PADDLE];
   [self loadSound:@"score" Slot:SOUND_SCORE];
}
```

Wczytywanie dźwięków za pomocą `AudioServicesCreateSystemSoundID` wymaga użycia `AudioServicesDisposeSystemSoundID` do usunięcia dźwięków po zakończeniu ich używania. Na początku metody `dealloc` dodaj więc kod odpowiedzialny za usunięcie każdego dźwięku z tablicy `sounds`.

```
// Usunięcie dźwięków.
for (int i = 0; i < 3; ++i)
{
   AudioServicesDisposeSystemSoundID(sounds[i]);
}
```

Zmodyfikuj również metodę `viewDidLoad`, aby dźwięki były wczytywane przed rozpoczęciem gry:

```
- (void)viewDidLoad
{
   [super viewDidLoad];
   [self initSounds];
   [self newGame];
}
```

Kolejnym krokiem jest dodanie prostej metody zajmującej się odtwarzaniem dźwięku ze wskazanego indeksu tablicy `sounds`. Odtwarzanie dźwięku następuje na skutek pojedynczego wywołania funkcji `AudioServicesPlaySystemSound()`. Wprawdzie to nie jest podstawą do utworzenia oddzielnej metody, z mojego doświadczenia wynika jednak, że implementacja obsługi dźwięku zmienia się podczas prac nad aplikacją. Modyfikacje będą łatwiejsze do wprowadzania, jeśli odtwarzaniem dźwięku zajmuje się oddzielna funkcja. Po metodzie `initSounds` umieść następujący kod:

```
- (void)playSound:(int)slot
{
   AudioServicesPlaySystemSound(sounds[slot]);
}
```

Zmodyfikuj także metodę `animate`, aby podczas uderzenia piłeczki o ścianę bądź paletkę był odtwarzany odpowiedni dźwięk. Przyznanie punktu graczowi również wiąże się z odtworzeniem efektu dźwiękowego:

```
- (void)animate
{
   // Przesunięcie piłeczki do nowego położenia z uwzględnieniem jej kierunku i prędkości.
   viewPuck.center = CGPointMake(viewPuck.center.x + dx*speed,
                                 viewPuck.center.y + dy*speed);

   // Sprawdzenie wystąpienia kolizji piłeczki z lewą lub prawą ścianą.
   if ([self checkPuckCollision:CGRectMake(-10,0,20,480)
                        DirX:fabs(dx)
                        DirY:0])
   {
      // Odtworzenie dźwięku uderzenia o ścianę.
      [self playSound:SOUND_WALL];
   }
   if ([self checkPuckCollision:CGRectMake(310,0,20,480)
                        DirX:-fabs(dx)
                        DirY:0])
   {
      // Odtworzenie dźwięku uderzenia o ścianę.
      [self playSound:SOUND_WALL];
   }
   // Wykrywanie kolizji piłeczki z paletkami graczy.
   if ([self checkPuckCollision:viewPaddle1.frame
                        DirX:(viewPuck.center.x - viewPaddle1.center.x) / 32.0
                        DirY:1])
```

```
    {
        // Odtworzenie dźwięku uderzenia o paletkę i zwiększenie prędkości piłeczki.
        [self increaseSpeed];
        [self playSound:SOUND_PADDLE];
    }

    if ([self checkPuckCollision:viewPaddle2.frame
                         DirX:(viewPuck.center.x - viewPaddle2.center.x) / 32.0
                         DirY:-1])
    {
        // Odtworzenie dźwięku uderzenia o paletkę i zwiększenie prędkości piłeczki.
        [self increaseSpeed];
        [self playSound:SOUND_PADDLE];
    }

    // Sprawdzenie, czy został zdobyty punkt.
    if ([self checkGoal])
    {
        // Odtworzenie dźwięku podczas dodawania punktu graczowi.
        [self playSound:SOUND_SCORE];
    }
}
```

Uruchom grę, odpowiednie dźwięki powinny być teraz generowane podczas uderzenia piłeczki o ścianę oraz gdy gracz zyskuje punkt. Informacje dotyczące nagrywania i edycji własnych dźwięków zostaną przedstawione w rozdziale 5.

Jeżeli nie słyszysz żadnych dźwięków, upewnij się, że urządzenie nie zostało wyciszone. Odpowiedni przełącznik znajduje się na górze po lewej stronie smartfona iPhone, tuż nad przyciskami regulacji głośności. Ponadto sprawdź poziom głośności ustawiony w urządzeniu. Jeżeli nadal masz problemy z odtworzeniem dźwięku, upewnij się, że pliki dźwiękowe zostały prawidłowo dodane do projektu oraz zainicjalizowane w kodzie.

Skorowidz

A

Adobe Fireworks, 82, 87–88
Adobe Illustrator, 84
Adobe Photoshop, 84
afconvert, 137
aktualizacje aplikacji, 183
animacja, 59, 124
animacja krążka, 130
archiwizacja aplikacji, 184
Audacity, 141
AudioToolbox, 75

B

biblioteka dźwięków, 138
biblioteka obiektów, 49
błędy, 56, 192

C

cena aplikacji, 190
certyfikat, 192
CPC, Cost Per Click, 199
Creative Commons Sample Plus, 138
CTR, 199
częstotliwość próbkowania, 136

D

Debug, 185
definicje stanów gracza, 157
dekodowanie sprzętowe, 138
dodawanie
  przycisków, 148
  obiektów do widoku, 35
  struktury AudioToolbox, 76
dyrektywa
  @end, 71
  @synthesize, 152
dźwięk, 74

E

edycja dźwięku, 141
edycja metadanych, 185
edytor WYSIWYG, 48
efekt
  dźwiękowy, 74, 135, 140
  Fade Out, 143
  Normalize, 143
ekran tytułowy, 97, 147, 154
Empty Application, 20
etykiety punktacji, 64

F

fala dźwiękowa, 135
filtrowanie nazw, 22
Fireworks, 82
format
  bezstratny, 141
  JPEG, 84
  PNG 24, 91, 93
  próbki, 136
  RGBA, 91
  uaktualnień, 183

formaty
obrazów, 84
plików dźwiękowych, 137
funkcja
animacji, 60
animate, 62
arc4random(), 40, 60
AudioServicesPlaySystemSound(), 77, 137
CGRectIntersectsRect, 62
checkGoal, 65
displayMessage, 67
exit(), 191
main(), 30
newGame, 67
NSLog(), 30
poprawiania kodu, 24
printf(), 30
rand(), 40
random(), 40
stop, 66
UIApplicationMain(), 30
funkcje symulatora, 26
funkcje trygonometryczne, 113

G

Garage Band, 138
generowanie liczb losowych, 40, 158
gest wstrząśnięcia, 72
Gimp, 82
gra
Air Hockey, 82
Glow Burst, 196
Lift Off, 195
matematyczna, 34
Paddles, 45, 106
Pong, 45
gracz-komputer, 154
gradient, 88
grafika
rastrowa, 83
wektorowa, 83

I

identyfikator aplikacji, Bundle ID, 181
ikony aplikacji, 102
informacje o wektorach i warstwach, 90
Inkscape, 82
inspektor atrybutów, 49
inspektor pliku, 25
instalacja Xcode, 17
Interface Builder, 48
interfejs użytkownika, 36
interfejs Xcode, 21

J

jakość dźwięku, 137

K

kierunek poruszania się piłeczki, 59
klasa
NSString, 65
Paddle, 109
Puck, 121
UIAlertView, 66, 67
kliknięcie, 199
kliknięcie reklamy, 198
kod rysujący prostokąty, 128
kod szesnastkowy koloru, 88
kolekcja efektów dźwiękowych, 138
kolizje, 62, 70, 127
kolor tła, 48
kompilator LLVM, 24
kompilowanie aplikacji, 26
komunikat błędu, 182
konwersja plików, 137
koszt, 199
koszt pobrania aplikacji, 199
krążek, 89
krok pośredni, 114

L

liczba losowa, 170
liczba pobrań aplikacji, 197
lista dozwolonych funkcji, 191
lista gier, 182
logika gry, 109

Ł

łącze do aplikacji, 193

M

Master-Detail Application, 20
menu kontekstowe, 38
menu Scheme, 26
metadane, 183
App URL, 184
Contact Email Address, 183

Copyright, 183
Description, 183
EULA, 184
Keywords, 183
Primary Category, 183
Rating, 184
Review Notes, 184
Secondary Category, 183
Support URL, 183
Uploads, 184
metoda
animate, 62, 113
checkGoal, 69, 130
computerAI, 155, 157, 165, 167
dealloc, 52
didFinishLaunchingWithOptions, 150
distance, 112
generate, 40
initWithView, 111
intValue, 41, 65
locationInView, 55
motionBegan, 73
motionCancelled, 73
motionEnd, 74
motionEnded, 73
move, 112
pause, 71
playGame, 152
reset, 61, 112
resume, 71
showTitle, 152
stringWithFormat, 40
submit, 40
terminate, 191
touchesBegan, 53, 116, 155
touchesCancelled, 53
touchesEnded, 53, 117
touchesMoved, 53
viewDidAppear, 41
viewDidLoad, 61, 131
viewDidUnload, 38, 52
metody
dotyku, 53
nieudokumentowane, 192
migawka, 103
moduł
Interface Builder, 33, 48, 105, 167
usuwania błędów, 53

N

nagrywanie dźwięku, 139
narzędzie afconvert, 137
nazwa aplikacji, 181
nazwa projektu, 103
normalizacja dźwięku, 140, 144
numer SKU, 181
numer wersji aplikacji, 182

O

obiekt
Paddle, 115
Puck, 126
UIAlertView, 41
UIImageView, 104
UIScreen, 86
Wall, 122
obraz
krążka, 90
paletki, 94
przycisku, 98
przycisku naciśniętego, 99
stołu, 94
obrazy wysokiej rozdzielczości, 103
obsługa
dotknięć, 57
dotyku, 119
paletek, 109
odrzucenie aplikacji, 187, 188
odsłona, 199
odświeżanie ekranu, 61
oferta, 198
okno
Image Preview, 91
inspektora pliku, File, 25
Organizer, 28, 186
OpenAL, 139
OpenGL Game, 19
opis aplikacji, 179
outlet, 36

P

Page-Based Application, 19
paletka, 91
panel
edytora, 23
modułu usuwania błędów, 26
narzędziowy, 25, 48
panel nawigacyjny, 21
dzienników zdarzeń, Log, 23
problemów, Issue, 23
punktów kontrolnych, Breakpoint, 23
symboli, Symbol, 22

panel nawigacyjny
usuwania błędów, Debug, 23
wyszukiwania, Search, 23
parametr seed, 40
pasek filtru, 22
pasek stanu, 48
plik
App Info, 47
background.png, 96
fireworks_puck.png, 91
GameAppDelegate.h, 25
GameAppDelegate.m, 30
GameViewController.h, 37
GameViewController.m, 38
GameViewController.xib, 33
Paddle.h, 109
main.m, 30
PaddlesAppDelegate.h, 150
PaddlesAppDelegate.m, 150
Paddles-Info.plist, 47
PaddlesViewController.h, 51, 115, 132
PaddlesViewController.m, 114, 152
PaddlesViewController.xib, 167
Puck.h, 128
Puck.m, 123
TitleViewController.xib, 170
pliki
.plist, 23
.png, 90
.xib, 23
dźwiękowe, 75, 137
graficzne Air Hockey, 100
interfejsu, 109
projektu, 29
rastrowe, 90
płyta audio CD, 136
pobranie, 199
polecenie importujące plik, 115
połączenie
obiektów z właściwościami, 105
outletu z kodem źródłowym, 51
portal
iOS Provisioning Portal, 181
iTunes Connect, 180, 186
poziom dźwięku, 140
poziom trudności, 69, 169
prefiks klasy, Class Prefix, 20
prędkość maksymalna, 113
prędkość piłeczki, 59
problem z paletkami, 118
proces zgłaszania aplikacji, 177
program
Adobe Fireworks, 82, 87–88
Adobe Illustrator, 84
Adobe Photoshop, 84
Audacity, 141
Fireworks, 82
Garage Band, 138
Gimp, 82
Inkscape, 82
Interface Builder, 48
projektowanie gracza, 154
promowanie aplikacji, 193
przeniesienie krążka, 160, 172
przezroczystość, 84
przycisk, 98
przyśpieszeniomierz, 72
punkt kontrolny, 25

R

ranking, 200
recenzje aplikacji iOS, 187, 196
rejestracja konta programisty, 17
rejestrowanie urządzenia, 28
reklama, 198
płatna, 198
w grze, 197
Release, 185
RGB, Red, Green, Blue, 88
rozdzielczość ekranu, 85
standardowa, 178
wysoka, 178
ruch krążka, 120
ruch paletek, 109

S

schemat
Debug, 185
Release, 185
serwisy społecznościowe, 195
Single View Application, 20
sklep
iTunes App Store, 17, 29, 194
Mac App Store, 17
sprzedaż, 200
stan
AI_BORED, 165, 173
AI_DEFENSE, 158
AI_OFFENSE, 162
AI_OFFENSE2, 171
AI_WAIT, 159, 163, 171
ataku, 162, 171
obrony, 158
stany aplikacji, 30
status First Responder, 41, 72

stół do gry, 95
strategia ataku, 163
struktura AudioToolbox, 75, 76
sygnał analogowy, 136
symbol projektu, 22
symulator iOS, 26
  ograniczenia, 27
  opcje, 27
System Audio Services, 74
system iOS 4, 71
szablon
  Empty Application, 20
  Master-Detail Application, 20
  OpenGL Game, 19
  Page-Based Application, 19
  Single View Application, 20
  Tabbed Application, 20
  Utility Application, 20
szablon projektu, 19
szybka pomoc, Quick Help, 25

T

Tabbed Application, 20
tablica sounds, 76
tarcie, 124
testy aplikacji, 186
tryb Retina, 106
tworzenie
  dźwięków, 138
  gracza komputerowego, 147
  grafiki, 81
  ikony, 100
  krążka, 87
  muzyki, 138
  outletu, 37
  paletki, 91
  połączenia, 37
  projektu, 102
  projektu Paddles, 46
  przycisków, 97
  stołu, 94
  winiety, 97
typ projektu, 19
typ wyliczeniowy, 165
tytuł aplikacji, 179

U

uaktualnianie aplikacji, 181
uruchamianie aplikacji, 150
urządzenie iOS, 28
usunięcie kontrolera widoku, 152
usuwanie dźwięku, 76
Utility Application, 20

W

Waiting for Upload, 193
wersja
  beta, 18
  Lite, 197
  systemu iOS, 46
widok, View, 49
winieta gry, 97
witryna
  Apple Developer, 17
  Google AdWords, 180
właściwość
  multipleTouchEnabled, 54
  viewPaddle1, 105, 110
wskaźnik CTR, 199
współczynnik konwersji, 199
wstrzymanie gry, 74
wybór słów kluczowych, 180
wyciek pamięci, 39
wykrywanie
  kolizji, 127, 130
  wstrząsu, 72
  zderzeń, 109
wyróżnienie aplikacji, 195
WYSIWYG, What You See Is What You Get, 48
wyświetlacz
  Multi-Touch, 53, 54
  Retina, 85
wyświetlanie klawiatury, 41
wyświetlanie komunikatu, 68

X

Xcode 4, 17

Z

zablokowane kontrolki, 192
zdarzenia dotyku, 57
zdarzenia ruchu, 72, 73
zerowanie gry, 69
zgłoszenie aplikacji, 188
zgłoszenie gry, 174
zrzuty ekranu, 177

Ż

żyroskop, 72

Nawet jeśli rynek gier i aplikacji mobilnych nie prześcignął jeszcze rynku tych produktów dla komputerów osobistych, to wkrótce to zrobi. Miliony użytkowników, przystępne ceny oraz powszechny dostęp do ekranów dotykowych, GPS i internetu — wszystko to sprawia, że gry na iPhone'a czy iPada są często o wiele bardziej atrakcyjne niż ich odpowiedniki przeznaczone na zwykłe komputery.

Najważniejszy jest pomysł na grę. Musi być nowatorski, atrakcyjny i wciągający. Jeśli już go masz, ten przewodnik pokaże Ci, w jaki sposób przejść do utworzenia prawdziwej i-aplikacji. Przedstawiony proces konstruowania rzeczywistej gry pozwoli Ci zdobyć podstawy dotyczące narzędzia Xcode i języka Objective-C. Jednocześnie nauczysz się, jak implementować logikę gry, przygotowywać grafikę i efekty dźwiękowe, a także opracować sztuczną inteligencję dla gracza-komputera. Już wkrótce możesz zdobyć sławę i czerpać z tego wymierne korzyści!

Z tą książką w ręku każdy programista zamieni pomysł na rzeczywisty produkt, gotowy do rozpowszechniania w sklepie iTunes App Store.

- Zacznij od prostej gry, wymagającej utworzenia jedynie około dwudziestu wierszy kodu.
- Zbuduj całkowicie od początku grę w hokeja na stole.
- Poznaj najlepsze praktyki w zakresie śledzenia wielu dotyków na ekranie.
- Stwórz animację i przygotuj funkcje wykrywania kolizji.
- Wykorzystaj niezbędne narzędzia do przygotowania atrakcyjnej grafiki dla gry.
- Dopracuj fizykę w grze, aby nadać jej większy realizm.
- Nagraj i przeprowadź edycję efektów dźwiękowych oraz utwórz własną muzykę odtwarzaną w tle.
- Stwórz projekt gracza-komputera grającego na różnych poziomach trudności.

Todd Moore założył firmę TMSOFT, by konstruować wyjątkowe aplikacje i gry przeznaczone dla smartfonów. Stworzona przez niego popularna gra Card Counter została wyróżniona przez Engadget, *Los Angeles Times* oraz CNET TV. Najpopularniejsza aplikacja Todda — White Noise — zdobyła uznanie iTunes App Store, *Health Magazine*, *Washington Post*, *PC Magazine* i *Late Night with Jimmy Fallon*.

ISBN 978-83-246-3965-6