Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Interaktivní multiuživatelská hra na plaformě iOS

Plzeň 2015 Nguyen Hoang Long

Zadani strana 1

Zadani strana 2

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2015

Nguyen Hoang Long

Abstract

This thesis is an overview of game development for platform iOS in framework Sprite Kit and programming language Swift. This thesis also contains brief information about basic frameworks for development of not only games but also independent application for iOS. Then it covers teoretical principles of framework Multipeer Connectivity. And the last part is the implementation of the game with Multipeer Connectivity and Sprite Kit in programming language Swift as an example. The reader should become acquainted with frameworks Sprite Kit and Multipeer Connectivity.

Abstrakt

Tato bakalářská práce se bude zabývat vývojem her pro platformu iOS ve frameworku Sprite Kit a programovacím jazyce Swift. Také obsahuje stručné informace ohledně základních frameworků pro vývoj nejen her, ale i nezávislých aplikací na platformu iOS. Dále obsahuje princip užití frameworku Multipeer Connectivity. Poslední částí této práce je ukázka implementace hry pomocí frameworků Multipeer Connectivity a Sprite Kitu. V této práci by se měl čtenář seznámit s frameworkem Sprite Kit a Multipeer Connectivity.

Obsah

Seznam obrázků

1 Úvod **1**

2 Operační systém iOS **2**
2.1 Architektura iOSu 2
2.1.1 Vrstva Cocoa Touch 3
2.1.2 Vrstva Media 3
2.1.3 Core Service 3
2.1.4 Core OS 3

3 Xcode **4**
3.1 LLVM překladač 4
3.2 LLDB debugger 4
3.3 iOS Simulator 5
3.4 Interface Builder 5
3.5 Jazyky 6
3.6 Licence 6

4 Objective-C a Swift **7**
4.1 Objective-C 7
4.2 Swift 7
4.3 Srovnání 7
4.3.1 Definice proměnných 8
4.3.2 Switch 8

5 Frameworky **10**
5.1 Foundation Kit 10
5.1.1 Základní třídy frameworku Foundation 10
5.2 UIKit 11
5.2.1 Třídy UIKitu 11

5.2.2 Elementy uživatelského rozhraní v UIKitu 12
5.2.3 IBOutlety a IBAction 13
5.3 AVFoundation 13
5.4 Sprite Kit 14
5.4.1 Herní cyklus 15
5.4.2 SKNode 16
5.4.3 SKScene 17
5.4.4 SKAction 17
5.4.5 Fyzikální svět 17
5.5 Další existující frameworky pro vývoj na platformě iOS 18

6 Engine pro vývoj iOS her **19**
6.1 Cocos2D-X 19

7 Multiuživatelské hry **20**
7.1 Klient-server 20
7.2 Multipeer Connectivity 20
7.3 Apple Game Center 20

8 Multipeer Connectivity **21**
8.1 Architektura 21
8.1.1 MCPeerID 21
8.1.2 MCNearbyServiceAdvertiser 22
8.1.3 MCNearbyServiceBrowser 22
8.1.4 MCSession 22

9 Core Animator **24**

10 Realizovaná aplikace **26**
10.1 MPCHandler 26
10.2 Obrazovky 31
10.2.1 SettingViewController 31
10.2.2 ViewController 34
10.3 GameViewController 37
10.4 GameScene 37
10.4.1 Vykreslení hry 37
10.4.2 Přidávání spritů 38
10.4.3 Fyzikální svět 39
10.4.4 Vykreslení scény s image sety 39
10.4.5 Kolize a doteky 40
10.4.6 Detekce dotyku 41

10.4.7 Ovládání hry 43
10.5 Testování aplikace 45
10.6 Instalace aplikace 46

11 Závěr **47**

Seznam obrázků

2.1 Vrsty v iOS 2

3.1 iOS Simulator 5

5.1 UIViewTable 12

5.2 Elementy UIKitu 12

5.3 Herní cyklus ve Sprite Kitu 15

5.4 Ukázka změny scén 16

9.1 Core Animator 24

9.2 Core Animator - export 25

10.1 Propojení obrazovek v ukázkové hře 31

10.2 SettingViewController 32

10.3 Seznam hráčů 35

10.4 Nastavení viditelnosti 35

10.5 Ukázka hry 38

10.6 Sada obrázku 39

11.1 Hlavni okno hry 56

11.2 Pozvání od hráče 57

11.3 Hra 57

1 Úvod

Cílem této práce je seznámit čtenáře s tvorbou interaktivních her na platformě iOS pro zařízení od společnosti Apple Inc. Poté prozkoumat existující frameworky, které se mohou hodit pro implementaci interaktivních her.

Dále je cílem ukázat, jakým způsobem by se daly vytvářet multiuživatelské hry, jaké jsou existující frameworky pro daný problém a jak multiuživatelské hry implementovat, především ve Sprite Kitu a jazyce Swift. Dále se budeme věnovat frameworku Multipeer Connectivity pro implementaci multiuživatelských her v jazyce Swift a Sprite Kitem.

Posledním úkolem je navrhnout multiuživatelskou hru pro platformu iOS, vytvořit navrženou hru a vyzkoušet její funkcionalitu na reálném zařízení. Navržená hra bude převážně implementována v jazce Swift pomocí frameworků Sprite Kit a UIKit, a komunikace mezi zařízením bude typu peer-to-peer za pomoci frameworku Multipeer Connectivity.

Po přečtení práce by čtenář měl mít představu o základních frameworcích a logice vývoje her pro platformu iOS, zejména logice fyzického světa ve Spirite Kitu napsaném v jazyce Swift.

2 Operační systém iOS

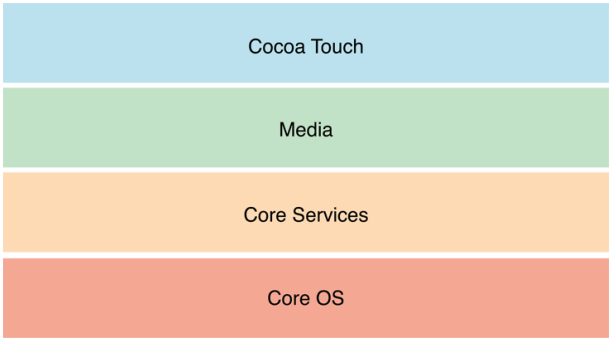
iOS je operační systém, který běží na zařízeních iPhone, iPod Touch, iPad a nove i na iWatch od společnosti *Apple Inc.* Operační systém je UNIXového typu tj. je multitaskingový a pro více uživatelů.

Operační systém iOS má podobnou architekturu jako operační systém OS X, který se používá pro počítače Mac, dalo by se říct, že iOS je miniaturní verze OS X. Viz [1].

K vývoji aplikací pro iOS je určeno rozhraní iOS SDK, kde se vyvíjí aplikace na základě frameworků v jazyce Objective-C, které jsou již přímo dostupné na fyzickém zařízení.

2.1 Architektura iOSu

Operační systém iOS je stejně jako operační systém OS X složen z několika systémových vrstev (viz obrázek 2.1. Obrázek převzat z [1]). Tyto vrstvy se nazývají **Cocoa Touch**, **Media**, **Core Service** a **Core OS**. Aplikace komunikuje se zařízením skrze několik nadefinovaných rozhraní, které jsou dostupné v podobě frameworků. Tyto frameworky jsou knihovny, hlavičkové soubory a další užitečné věci důležité pro vývoj aplikací.



Obrázek 2.1: Vrsty v iOS

2.1.1 Vrstva Cocoa Touch

Cocoa Touch je klíčová vrstva, která určuje vzhled aplikace z pohledu uživatele. Tato vrstva využívá framework **UIKit**, který nabízí základní funkce jako jsou uživatelská rozhraní, notifikace, výběr textu, kopírování, dotyky apod.

Vrstva **Cocoa Touch** dále zprostředkovává rozpoznání gest a komunikaci s blízkými zařízeními přes Bluetooth. Technologie této vrstvy je napsaná v programovacím jazyce Objective-C.

2.1.2 Vrstva Media

Tato vrstva zajišťuje práci s grafikou, zvukem a videosoubory. Tato vrstva zahrnuje i funkce, které podporují 3D animace, vibrace, přehrávání a nahrávání audio záznamů a další.

2.1.3 Core Service

Tato vrstva zajišťuje systémové služby pro aplikace, které jsou většinou založeny na databázi. Vrstva dále poskytuje podporu SQL a XML. SQL databáze je možné vkládat do aplikací bez nutnosti přístupu ke vzdálenému serveru a lze vytvářet lokální databáze v aplikaci.

2.1.4 Core OS

Tato vrstva obsahuje nízkoúrovňové funkce, na kterých jsou navrženy ostatní technologie. Například zpracování obrázku, digitálních signálu, zabezpečení služeb, apod.

3 Xcode

Xcode je vývojové prostředí od společnosti Apple pro vývoj softwaru na operační systém OS X a iOS. **Xcode** je freewarový opensource zdarma ke stažení na App Storu, ale pouze pro operační sytémy OS X. Sestava obsahuje v základu Xcode IDE, kompilátor pro jazyky Swift a Objective-C, nejnovější verzi OS X a iOS SDK, iOS simulátor a mnoho dalších základních nástrojů.

3.1 LLVM překladač

Xcode v původních verzí používal modifikovaný překladač GCC (GNU Compiler Collection), ale v řadě Xcode 3.2 byl představen překladač LLVM (původní jméno Low Level Virtual Machine), který se v projektech používal spolu s překladačem GCC. V řade Xcode 5.0 se Apple uchýlil pouze k používání LLVM překladače.

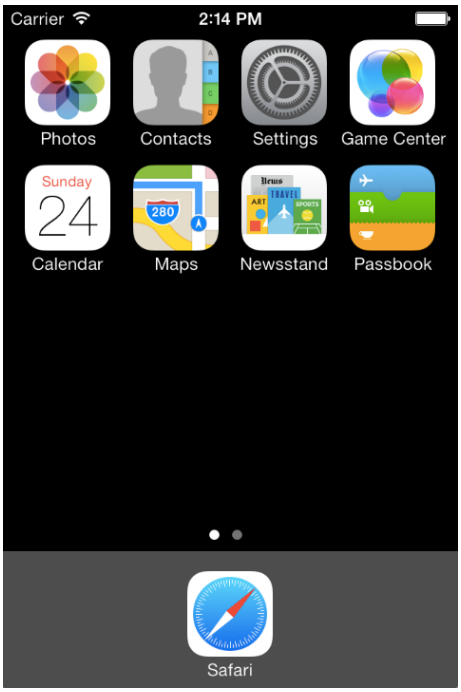
LLVM překladač je založen na opensourcovém projektu LLVM.org. Projekt LLVM začal v roce 2000 na státní univerzitě University of Illinois at Urbana-Champaign pod vedením Vikrama Adveho a Chrise Lattnera. LLVM byla původně infrastruktura pro výzkum dynamické kompilace. V roce 2005 najala společnost Apple celý vývojářský tým, který se podílel na vývoji LLVM pro vývoj jejich systémů. Dle [13] je LLVM název celkového projektu, nikoliv zkratka. A navzdory jeho jménu nemá skoro nic společného s virtuálními stroji.

3.2 LLDB debugger

Součástí Xcodu je i vysokovýkonný debugger LLDB, který byl vyvíjen spolu s LLVM kompilátorem. Stejně jako LLVM kompilátor, je sestaven z několika sad komponent a je kompatibilní s již existujícími knihovnami LLVM kompilátoru.

3.3 iOS Simulator

Xcode dovoluje spouštět a testovat aplikace na fyzickém zařízení, ale pouze v případě, že máte zaplacený vývojářský účet v Apple Developer programu nebo v univerzitním Apple Developer programu. V opačném případě máte možnost testovat svou aplikaci na iOS simulátoru, která je zdarma součástí Xcodu. Aplikace umožňuje simulovat jak iPhone tak i iPad zařízení. Simulace umožňuje využívat většinu vlastností jako na reálném zařízení spolu s identickým uživatelským rozhraním viz obrázek 3.1.



Obrázek 3.1: iOS Simulator

3.4 Interface Builder

Další součástí Xcodu je Interface Builder, aplikace, která umožňuje vytvářet uživatelské rozhraní bez psaní jakkéhokoliv kódu. Aplikace je založena na modelu MVC (Model-view-controller). Interface Builder umožňuje přidávat tlačítka, pohledy, propojení pohledů a mnoho dalšího.

3.5 Jazyky

Nativním jazykem operačního systému iOS je Objective-C, nyní nově i jazyk Swift. V tomto jazyce je napsáno Cocoa Touch API, který tvoří základ pro každou aplikaci na platformě iOS. Vývoj aplikací pro platformu iOS není omezen jen na tyto dva jazyky, v oficiálním prohlášení [11] je, že Xcode podporuje i další jazyky jako jsou C, C++, Objective-C++, Java, AppleScript, Python, Ruby, Rez, GNU Pascal, Free Pascal, Ada, C#, Perl a D.

3.6 Licence

Xcode je opensourcový freeware volně k dispozici ke stažení. Pro běžného uživatele je povoleno vytvářet projekty a spouštět je na iOS simulátoru. Ale v případě, že chcete spustit aplikaci na fyzickém zařízení, či využívat služby iTunnesConnect, musíte mít vývojářský účet v Apple Developer programu, za který se ročně platí nemalá částka (pro běžné uživatele 99$ a pro firmy 299$). K dispozici je také univerzitní účet v rámci iOS Developer University Program, který je zdarma.

4 Objective-C a Swift

4.1 Objective-C

Objective-C je objektově orientovaný jazyk, který je založen na jazyce C rozšířený o systém zasílání zpráv z jazyka Smalltalk. Objective-C byl vyvinut Bradem Coxem a Tomem Lovem v roce 1986 ve společnosti Stepstone. Původně byl vyvinut jako hlavní programovací jazyk pro počítace NeXT s operačním systémem NeXTSTEP, které se dnes již nevyrábí. Přesto, že se počítače NeXT už se nevyrábějí, hlavní myšlenka softwarového prostředí přetrvala ve standardu OpenStep a díky tomu se poté stal hlavním programovacím jazykem společnosti Apple, jejíž operační systémy začali využívat OpenStep. Překladač tohoto jazyka je součástí GCC, ale současné době, se využívá clang, který je součástí sad v LLVM.

4.2 Swift

Swift je nový multi-paradigmatický, kompilovaný programovací jazyk pro vývoj OS X a iOS aplikací distribuovaná firmou Apple Inc.,jejímž základem jsou nejlepší aspekty, převzaté z jazyků Objective-C, Rust, Haskell, Ruby, Python, C#, CLU a mnoha dalších. Vývoj Swiftu započal Chris Lattner roku 2010 a následně, v roce 2014, jej zveřejnil. Swift je obdobou jazyka Objective-C s lepší syntaxí a modernějšími koncepty, takže nedovoluje tolik chyb programátora jako je u Objective-C. Při jeho uvedení byl představen jako „Objective-C without the C (Objective-C bez C)“[12], protože na rozdíl od Objective-C není nutné používat pointery(ukazatele).

4.3 Srovnání

Výhodou Swiftu oproti Objective-C je ten, že Swift je nový a modernější jazyk. Je vytvořen především na základě vlastností jiných jazyků. Oproti Objective-C není za potřebí používat ukazatele a volání metod je nahrazen tečkovou anotací, tudíž je přehlednější. Dále u Swiftu není za potřebí ukon-

čovat příkazy středníkem, nejsou vyžadovány hlavičkové soubory, má plnou podporu Unicodu a typovou kontrolu.

4.3.1 Definice proměnných

Listing 4.1: Definice proměnných v Objective-C

```
NSString *user = @"Steve";
int days = 5;
NSString *s =
[NSString stringWithFormat:@"posted by %@ (%d days ago)",
 user, days];
```

Listing 4.2: Definice proměnných ve Swiftu

```
var user = "Steve"
var days = 5
var s = "posted by \(user) \(days) ago"
```

Jak je z příkladů 4.1 a 4.2 vidět, tak u Swiftu není zapotřebí uvádět typ proměnné, pokud je vynechán typ proměnný, tak se automaticky nastaví výchozí typ. Pokud chcete Swiftu zadat typ proměnný, bylo by to například „var days: Int? = 5“, v takovém případě to znamená, že v dané proměnné smí být pouze Integer a otazník u typu udává, že proměnná může být nil (null), pokud by místo otazníků byl vykřičník, znamenalo by to, že proměnná musí být definovaná tj. nesmí být null.

4.3.2 Switch

Další hodně silnou stránkou jazyka Swift je konstrukce Switch. Je nutné mít na paměti to, že u Objective-C není možné použít Switch pro datové typy typu String.Pro rozřazení dle typy String v Objective-C se musí použít podmínky if.

Listing 4.3: Switch v Objective-C

```
if ([osoba.povolani isEqualToString:@"policista"]) {
  NSLog(@"Polda");
} else if ([osoba.povolani isEqualToString:@"bandita"]) {
  NSLog(@"Bandita");
} else {
  NSLog(@"UFO");
}
```

Listing 4.4: Switch ve Swiftu

```
switch osoba.povolani {
  case "policista":
    println("Polda")
  case "Bandita":
    println("Bandita")
  default:
    println("UFO")
}
```

Jak bylo řečeno předtím, Switch u Objective-C funguje jen pro jedno číslo např. „case 1: NSLog(@"jedna");break;“, ale pouhé porovnání Stringů není u Swiftu pouhou silnou stránkou, co dělá Swiftem silným je následná práce Switche s čísly:

Listing 4.5: Swift ve Swiftu

```
switch cislo {
case 0, 1, 2:
    println("Male")
case 3...7:
    println("Stredni")
case 8..<10:
    println("Velke")
case _ where cislo % 2 == 0:
    println("Sude")
case _ where cislo % 2 == 1:
    println("Liche")
default:
    break
}
```

5 Frameworky

5.1 Foundation Kit

Framework Foundation je základní knihovnou pro jazyk Objective-C pro vývoj aplikací na platformě iOS. Poskytuje programátorovi několik základních obalových tříd pro ukládání dat jako jsou objekty, stringy, čísla, slovníky atp. Třídy tohoto frameworků využívají prefix „NS“ (převzato od **NeXTSTEP**, či **NeXT/Sun**).

Základní třídy ve **Foundation** jsou podobné jako ve Frameworku **UIKit**, který je popsán níže v sekci 5.2. Ačkoliv mají tyto dva frameworky podobné třídy, tak jsou zcela odlišné, zatímco **UIKit** nám pomáhá vytvářet a starat se o elementy uživatelského rozhraní, tak u Foundation se staráme spíše z pohledu kódu. Jinak řečeno, Foundation nám poskytuje třídy pro práci s numerickými hodnotami, řetězci, kolekcemi a mnohým dalším.

5.1.1 Základní třídy frameworku Foundation

- **NSObject:** Jak už z názvu vyplývá, tak v Objective-C všechny objekty dědí od třídy NSObject.
- **NSString:** Tato třída se využívá pro manipulaci se Stringovými datovými typy, stringy jsou reprezentovány v Unicodu.
- **NSValue a NSNumber:** Třída NSValue je obalový kontejner pro datové objekty. Třída NSNumber je podtřídou NSValue, která se stará o numerické datové typy jako jsou Interger, Double a Float.
- **NSArray:** Třída NSArray slouží pro práci se seznamem objektů. Do třída NSArray se dají ukládat pouze objekty, skalarní datové typy jako ukazatele a struktury vkládat nelze.

5.2 UIKit

Jeden z nejužitečnějších frameworků je **UIKit** - poskytuje všechny potřebné prvky uživatelského rozhraní pro vývoj kvalitního a rychlého vykreslování UI, které lze snadno ovládat pomocí funkcí a struktur.

UIKit je objektově orientovaný framework, který má na starost většinu věcí ohledně uživatelského rozhraní. **UIKit** je nejvíce užívaným frameworkem pro vývoj aplikací pro platformu iOS, protože se s ním velice snadno definují základní komponenty aplikace jako jsou labely, tlačítka, tabule, navigační panely apod. Tento framework je založen na modelu Model-View-Controller.

Pro design uživatelského rozhraní s **UIKitem** je doporučeno používat soubor zvaný **Storyboard**, který je již v Xcodu. Storyboard poskytuje model pohledu na aplikaci a požívá se k návrhu uživatelského rozhraní na platformě iOS. Nahrazuje původní xib soubory, které se používaly dříve. Soubory lze editovat v nástroji **Interface Builder** a data jsou ukládána jako XML.

5.2.1 Třídy UIKitu

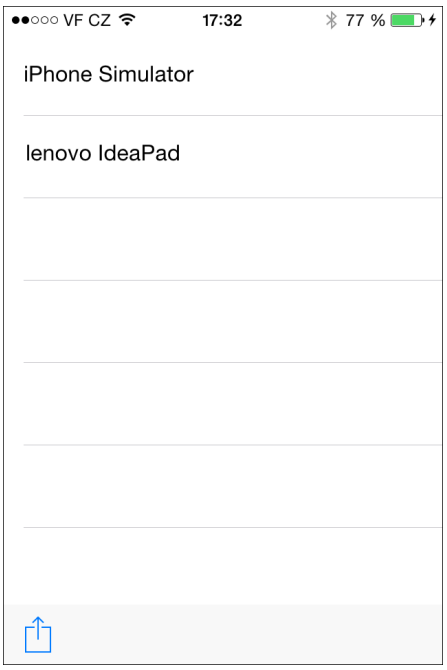
UIKit má mnoho dalších tříd, ale pro vývoj jsou tyto nejdůležitější a nejvíce užívané:

- **UIApplication:** Třída UIApplication slouží jako centrální bod pro koordinaci a řízení každé aplikace pro platformu iOS. Ačkoliv je tato třída velice důležitá, ale málo kdy se s ní budete přímo stýkat a implementovat.
- **UIViewController:** Jsou jednotlivé pohledy aplikace. Při vývoji aplikace je každý pohled aplikace obalená třídou UIViewController, ve kterým se definují i layouty a propojují se pohledy při více pohledech aplikace.
- **UIView:** V každém UIViewControlleru musí obsahovat komponentu UIView, aby se do pohledu daly dále přidávat objekty typu tlačítek, popisek, mřížek atp.
- **UIGestureRecognizer:** Je abstraktní třída, která má na starosti doteky na obrazovce typu kliknutí, táhnutí, dlouhé stlačení atd.

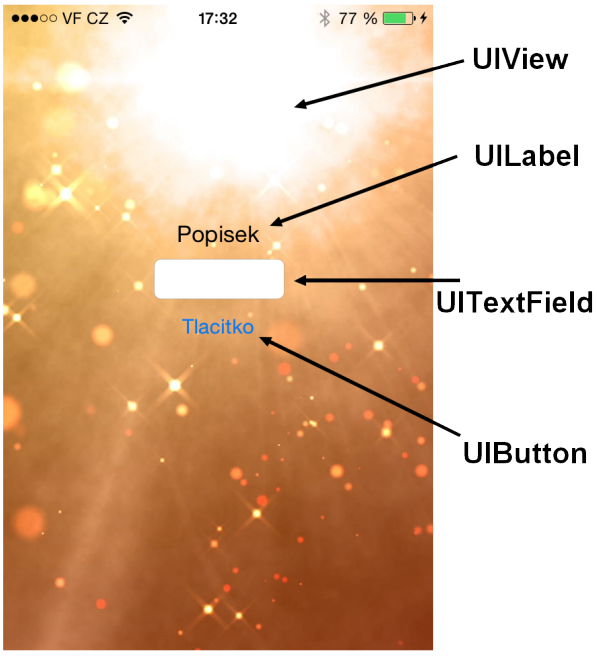
5.2.2 Elementy uživatelského rozhraní v UIKitu

Kromě tříd převzatých od **Foundation**, má **UIKit** mnoho dalších, ale pro vývoj jsou tyto nejdůležitější a nejvíce užívané:

- **UITableView:** Je pohled, který vytváří seznam a zobrazuje na obrazovku seznam položek v řádcích. Na jednotlivé položky seznamu je možné klikat a v případě, že položky mají nastavený IBAction, tak provedou akci. Ukázka UITableView je níže v obrázku 5.1.
- **UILabel:** Element statického textu resp. popisku. Viz obrázek 5.2.
- **UIButton:** Tlačítka aplikace. Většinou se tlačítka nastavují jako IBAction, takže při stisknutí provedou akci. Viz obrázek 5.2.
- **UITextField:** Editovatelné textové pole aplikace. Při stistknutí na pole vyjede klávesnice. UITextField se většinou nastavují jako IBOutlet, který sám o sobě nic nedělá, je to jenom proměnná, ke které můžeme přistupovat, pokud chceme dostat text z pole. Viz obrázek 5.2.



Obrázek 5.1: UIViewTable



Obrázek 5.2: Elementy UIKitu

5.2.3 IBOutlety a IBAction

IBOutlet a **IBAction** slouží k propojení uživatelského rozhraní(tlačítka, labely apod.) s kódem tj. propojení **Interface Builderu** s třídou pohledu na obrazovku.

- **IBOutlet:** Přidává se k elementům, které neprovádí nějakou činnost. Slouží k tomu, aby ty elementy **Interface Builder** viděl, jinak řečeno, abychom jim mohli programově přidávat parametry apod. V obrázku 5.2 máme vytvořený jeden popisek jako **UILabel** a jedno textové pole jako **UITextField** a v obrázku 5.1 jsme si oba dva elementy inicializovali v kódu a pojmenovali. Díky tomu s nimi můžeme pracovat jako například, že se podívíme, co je v textovém poli napsáno s kódem „textovePole.text“. U outletů se musejí deklarovat zda jsou weak nebo strong, ale mezi strong a weak není žádný rozdíl kromě priority.

Listing 5.1: Ukázka použití IBOutlet

```
@IBOutlet weak var textovePole: UITextField!
@IBOutlet strong var popisek: UILabel!
```

- **IBAction:** se může volat u jakkéhokoliv elemetu, který se provede při stisknutí. V 5.2 máme vytvořené tlačítko a při stisknutí tlačítka zobrazí hláška s potvrzením ok, které nic nedělá.

Listing 5.2: Ukázka použití IBAction

```
@IBAction func tlacitko(sender: AnyObject){
        let okAlert = UIAlertAction(title: ''Ok'',
                style: UIAlertActionStyle.Default){
                (alertAction) -> Void in
        }
}
```

5.3 AVFoundation

AVFoundation je framework, který umožňuje jazyku Objective-C spouštět audio-vizuální média a prací s nimi jako je překódování, úpravy, či zachytávání obrazů z kamery a mikrofonu.

Listing 5.3: Ukázka použití AVFoundation

```
import AVFoundation

class GameScene: SKScene {

    // inicializace cesty ke zvuku, ktery mame v projektu
    var coinSound =
    NSURL(fileURLWithPath:
        NSBundle.mainBundle().pathForResource
        ("coin", ofType: "wav"))

    // inicializace audio playeru
    var audioPlayer = AVAudioPlayer()

    // pridani do sceny a spusteni
    override func didMoveToView(view: SKView) {
        audioPlayer =
        AVAudioPlayer(contentsOfURL: coinSound, error: nil)
        audioPlayer.prepareToPlay()
        audioPlayer.play()
    }

}
```

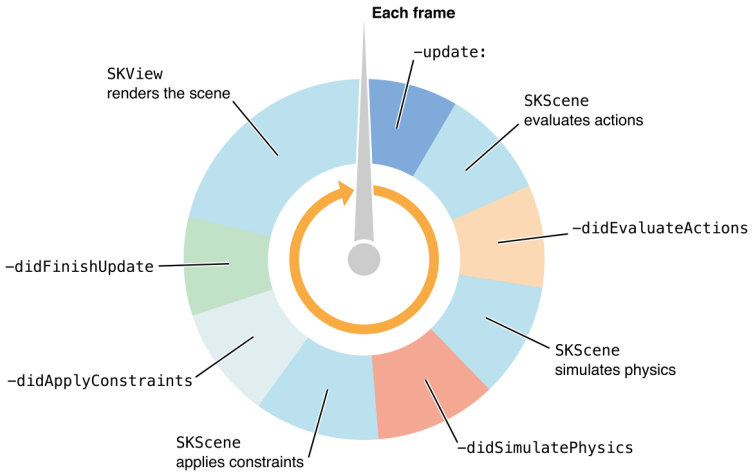
5.4 Sprite Kit

Sprite Kit je nový framework pro vývoj 2D her pro platformu iOS. **Sprite Kit** zajišťuje vykreslování grafiky a animace. Také dovoluje vytvářet komplexní speciální efekty, simulaci fyzikálních zákonů a podporu audio médií. Je to framework, u kterého se uvádí, že vývoj her v tomto frameworku je zcela optimalizovaný pro zařízení od společnosti Apple Inc. **Sprite Kit** využívá grafický hardware na fyzickém zařízení, aby zobrazil 2D obrázky ve vysoké snímkové frekvenci. **Sprite Kit** dále podporuje funkce jako jsou pouštění hudby, renderování textů apod. a navíc, je již zabudovám v Xcodu.

5.4.1 Herní cyklus

Na konci každého herního cyklu se vykreslí objekty na scénu, a to vše se děje v zákulisí, kde si **Sprite Kit** vynucuje vykreslení u **OpenGL**. Vzhledem k tomu, že se **Sprite Kit** snaží co nejrychleji vykreslovat rychlostí 60 snímků za vteřinu, tak mohou nastat problémy s poklesem rychlostí vykreslování snímku, když je metoda update zatěžována složitými algoritmy.

V každé scéně **Sprite Kitu** probíhá nekonečný cyklus, který můžete vidět na obrázku 5.3. Jeden herní cyklus je jeden snímek scény, kde u her se typicky vykresluje rychlostí 30 či 60 snímků za vteřinu.



Obrázek 5.3: Herní cyklus ve Sprite Kitu

Každý snímek ve **Sprite Kitu**, jak je vidět z obrázku 5.3 (obrázek převzat z [2]) dělá následující věci:

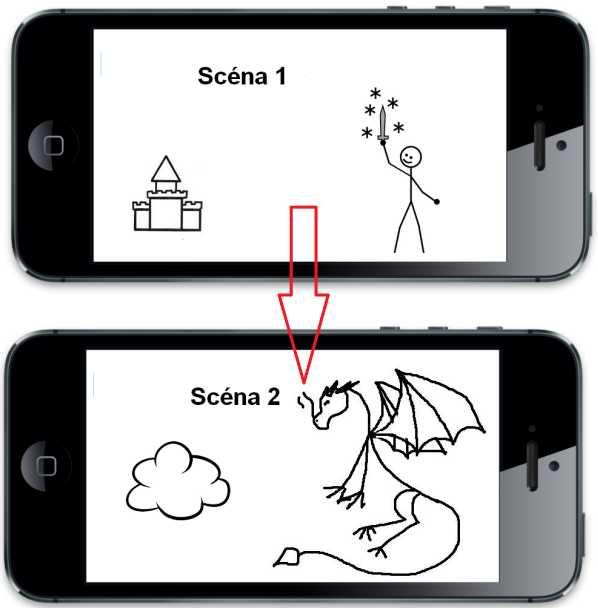
- **update:** Metoda, která se volá v každém snímku. V této metodě je vhodné posouvat, či otáčet sprity a kontrolovat výhru nebo prohru.
- **didEvaluateActions:** Ve **Sprite Kitu** je mnoho akcí, jako jsou rotace, pozicování spritů, detekce kolizí apod. a tato metoda provádí všechny akce v závislosti na nové pozici spritu.
- **didSimulatePhysics:** Ve **Sprite Kitu** platí fyzikální zákony jako je gravitace a dynamika při nárazu objektů, a to tato metoda zprostředkovává.

5.4.2 SKNode

Každý objekt na obrazovce ve **Sprite Kitu** pochází ze třídy zvané **SKNode**. **SKNode** samo o sobě nic nevykresluje, spíše slouží pro inicializaci elemetů, které se budou přidávat na scénu.

Pár běžně používaných elemetů, které jsou ve **SKNode** k dispozici:

- **SKSpriteNode:** Nejvíce využívaný node pro vývoj her, který dovoluje vykreslovat sprity jako dvourozměrný obrázek. Sprity mají vetšinou svůj název a reagují na **SKAction**, což jsou například dotyky apod.
- **SKLabelNode:** Tento node se používá pro vykreslování textů v jedné řádce, které se chovají jako sprite, tudíž na ně paltí **SKAction**. Tento node podporuje všechny fonty, i vlastní vytvořené.
- **SKEmitterNode:** Tento node se používá k vyzařování částic, neboli speciálních efektů. Například, když jeden sprite narazí do druhého, začnou vyzařovat jiskry.



Obrázek 5.4: Ukázka změny scén

5.4.3 SKScene

SpriteKit je knihovna umožňující vykreslování grafiky a animací do scén. Texturám ve scéně se nazývá sprite a můžeme se na ně dívat jako jednotlivé postavičky či objekty, co se nachází na scéně jeviště tj. z obrázků 5.4 ve scéně 2 jsou „sprity“ drak, mráček a nápis scény.

Jenom jedna scéna může být v jednom čase aktivní, a každá scéna je prezentována jako **SKView**. Scéna funguje jako obří plátno x, y, z souřadnicového systému, kam se jednotlivé sprity vykreslují jako objekt, kde sprite provádí činnost k dané scéně. Během jedné scény sprity vykonávají nějakou činnost, jako je pohyb dopředu nebo skoky, při doteku uživatele na sprite.

5.4.4 SKAction

SKAction zajišťuje animace pro sprity a přehrávání zvukových efektů. Umožňuje spritům provádět rotace a posuny jejich pozic.

Běžné **SKAction**, co jsou k dispozici:

- **Move actions:** Tato akce slouží k animacím posunu sprite. V této akci jsou na výběr různé způsoby, jako je lineární posun, zrychlený posun, zpomalený posun atd.
- **Sequence action:** Tato akce slouží k zřetězení akcí, jednoduše, každá akce spouští nějakou další akci.
- **Repeat actions:** Tato akce dovoluje opakovaně provádět nějakou činnost donekonečna nebo jen nějaký určitý počet.
- **Sound action:** Tato akce dovoluje spouštět zvukové efekty. Ale je dobré mít na paměti, že je jedno, v jakým spritu se zvuk pustí, bude stejně ve scéně slyšet.

5.4.5 Fyzikální svět

Ve **Sprite Kitu** se všechno ve scéně chová jako dynamický svět plný vektorů, ve kterém platí všechny zákony fyziky z reálného světa. Každý sprite je

předmět, či stvoření, které má fyzické tělo, na které platí gravitace. Síla gravitace a směr je určena podle směrového vektoru a délky směrového vektoru. Defaultně je směr gravitace nastavena směrem dolů. Sprity spolu kolidují, a na základě jejich definovaných těl se mohou od sebe odrážet, či rozbíjet.

5.5 Další existující frameworky pro vývoj na platformě iOS

Pro vývoj aplikací na platformě iOS existuje nespočet frameworků viz [15], zde je jich pár vyjmenováno:

- **CloudKit:** Slouží k stahování a nahrávání dat na iCloud.
- **EventKit:** Slouží pro práci s kalendářem.
- **MapKit:** Pro práci s mapovým rozhraním pro vaší aplikaci.
- **NotificationCenter:** Pro implementaci notifikací a widgetů.
- **PassKit:** Framework pro tvorbu digitálních jízdenek do autobusů, lístků na akci, apod.
- **Twitter:** Pro propojení vaší aplikace se sociální sítí Twitter.

6 Engine pro vývoj iOS her

Jonathan Blow [8] se zmínil o tom, že v poslední době je vývoj her čím dál, tím složitější, a že hlavní výzvou je dostat kód do nějaké finální podoby. K tomu je zapotřebí znát širokou škálu algoritmů a stojí hodně úsilí, dát něco takového dohromady.

Předtím bylo možné vytvářet hru pro iOS jen v enginu **OpenGL ES**, na kterém dodnes pracuje **Sprite Kit**. API je optimalizován pro užití 2D a 3D grafiky pro zařízení od společnosti Apple. Ale pro začínající programátory byl **OpenGL ES** velkou překážkou, proto vznikly frameworky třetích stran jako jsou **Cocos2D**, **Sparrow**, **Corona**, nebo **Unity**, které měli ulehčovat práci s **OpenGL**. Jelikož tyto frameworky nebyly od Applu nebo podporovány Applem, tak nastával problém při každém vydání nového operačního systému iOS. Aby se tento problém s kompatibilitou vyřešil, vydal Apple nový framework zvaný **Sprite Kit**.

6.1 Cocos2D-X

Cocos2D-X je multiplatformní opensourcový engine ze třetí strany pro vývoj 2D her. Je to jeden z oblíbených frameworků pro vývoj 2D mobilních her a je zdarma. Jeho výhodami jsou rychlost, stabilita, jednoduchost a velká komunita. Dále má utilitu, která umožňuje multiplatformu. Vývoj her lze provádět v Xcode v Objective-C a za pomocí utility SpriteBuilder Android Plugin bude aplikace podporována na iOSu, Macu a Androidu.

Logika **Sprite Kitu** se scénami, sprity a akcemi je velice podobná **Cocos2D-X**, proto pro uživatele, co mají zkušenosti s **Cocos2D-X** by **Sprite Kit** neměl být žádný problém.

7 Multiuživatelské hry

K vytvoření multiuživatelské hry na platformu iOS jsou 3 možnosti:

7.1 Klient-server

Klasicky server s klientem, které spolu komunikují TCP či UDP protokolem a hra je hostovaná na vzdáleném serveru. Pro platformu se iOS se dá psát server v jazyce C# nebo Objective-C.

7.2 Multipeer Connectivity

Další možnost je **Multipeer Connectivity** framework, což je oficiální framework od společnsti Apple Inc., která umožňuje připojení peer-to-peer přes Wi-Fi a nebo osobní síť Bluetooth.

7.3 Apple Game Center

Další rozvoj multiuživatelských aplikací je možný v oblasti online služeb **Apple Game Center**, které slouží k porovnání achievmentů, vyhledávání soupeře, či tvoření achievmentů. Ale k této službě je zapotřebí placený developerský účet tj. tato služba není zadarmo.

8 Multipeer Connectivity

Multipeer Connectivity je nový framework představený společností Apple. **Multipeer Connectivity** je framework, který detekuje fyzická zařízení v blízkosti a zajišťuje spojení mezi nimi pomocí osobní bezdrátové sítě přes Wi-Fi nebo Bluetooth.

Funkce **Multipeer Connectivity** by se dala rozdělit do několika fází: vysílání, hledání, spojení a výměna dat. Jednoduše, máme jedno zařízení, které chce, aby ho ostatní viděli, tak začne vysílat(advertising) o své službě, že chce být nalezeno. Druhé zařízení zapne svůj vyhledávač(browser) a hledá svou danou službu. Pokud se vysílaná služba a hledaná služba shodují, uvidí browser advertisera a případně se na něj připojí. A nyní jsou naše obě zařízení propojena, tak může začít výměna dat. V **Multipeer Connectivity** může probíhat výměna dat třemi způsoby, prvním je jako objekt instance **NSData**, druhým způsobem jsou soubory nebo dokumenty a třetím způsobem je streamování dat.

8.1 Architektura

Multipeer Connectivity se skládá z několika hlavních částí: **MCSession**, **MCNearbyServiceAdvertiser**, **MCNearbyServiceBrowser** a **MCPeerID**.

8.1.1 MCPeerID

MCPeerID je unikátní identifikátor zařízení, neboli uživatele, aby se rozlišila zařízení v blízkosti. Většinou je to hned první věc, co musí být inicializována, tato třída má proměnou displayName, což je jméno, které se bude ostatním zařízením zobrazovat.

8.1.2 MCNearbyServiceAdvertiser

MCNearbyServiceAdvertiser je třída, která vysílá zařízením v blízkosti o své službě. Při inicializaci se musí určit serviceType jak je vidět v kódu u 8.1, serviceType nesmí být delší než 15 znaků a musí obsahovat pouze malá písměna ASCII znaků. V případě 8.1 vysílá zařízení o sobě, že je služby „sluzbaboty“.

Listing 8.1: Ukázka použití Advertiseru

```
var advertiser: MCNearbyServiceAdvertiser!

init(){
  advertiser = MCNearbyServiceAdvertiser(peer: peer,
    serviceType: ''sluzbaboty'')
}
```

8.1.3 MCNearbyServiceBrowser

MCNearbyServiceBrowser zajišťuje vyhledávání blízkých zařízení, které vysílají stejnou službu jako kterou browser vyhledává. Viz kód 8.2. Ve třídě browseru je implementována ještě jedna funkce, při zavolání metody setupBrowser(), vyskočí obrazovka v **UIView**, která zobrazuje zařízení v okolí a ke kterým jsme připojeni.

Listing 8.2: Ukázka použití Browseru

```
var browser: MCNearbyServiceBrowser!

init(){
  browser = MCNearbyServiceBrowser(peer: peer,
    serviceType: ''sluzbaboty'')
}
```

8.1.4 MCSession

MCSession zajišťuje a udržuje spojení mezi zařízeními, má a detekuje tři stavy: nepřipojen, připojuje se a připojen. Dále **MCSession** má dva typy

odesílání dat Reliable a Unrealiable. U Realiable je údajně zaručeno, že data vždycky přijdou, a samozřejmě, u Unrealiabe nemusí vždy přijít viz ukázka odesílání a příjmu dat v 8.3.

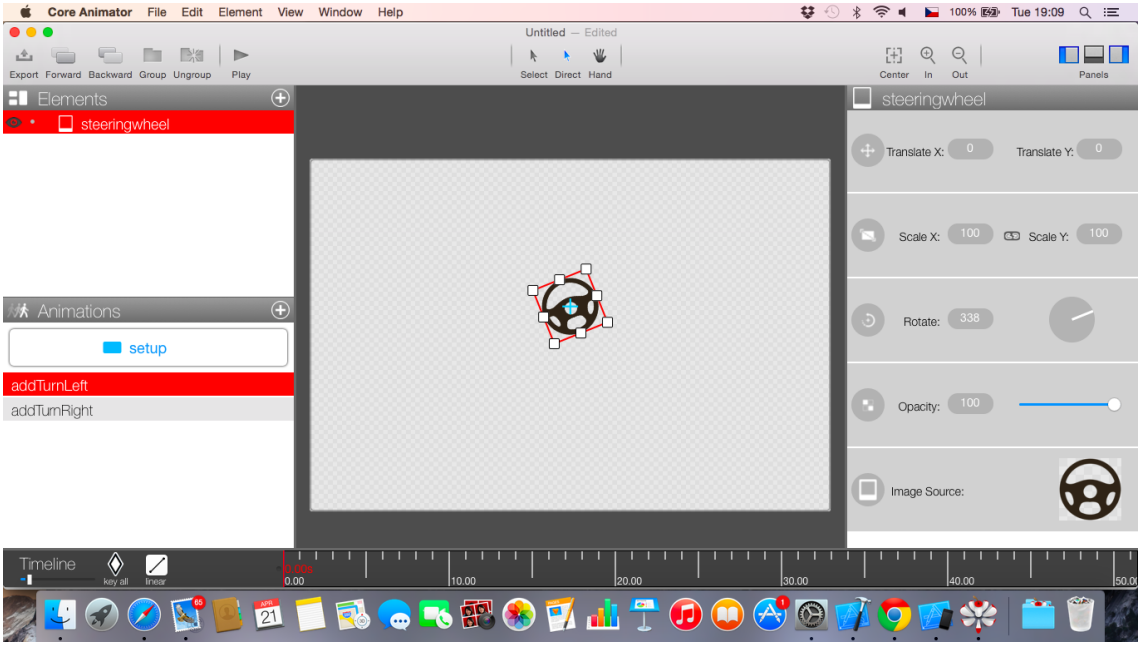
Listing 8.3: Ukázka odesilani a příjmu dat

```
var session: MCSession!

//inicializace
init(){
  session = MCSession(peer: peer)
}

//odelani dat
let dataKPoslani =
NSKeyedArchiver.archiveDataWithRootObject(odesilanaData)

session.sendData(dataKPoslani, toPeers: nepritelovi,
withMode: MCSessionSendDataMode.Reliable, error: &error)

//prijem dat
func session(session: MCSession!,
didReceivedData data: NSData!, fromPeer peerID: MCPeerID!){

        println(data)

}
```

9 Core Animator

Core Animator je moderní grafické studio pro práci se sprity k tvorbě jednoduchých animací. Je to placený software, který je dostupný na platformách OS X a OS X Yosemite. Core Animator je skvělá pomůcka pro tvorbu animací k vaší aplikaci, protože animaci, u níž byste strávili několik hodin psaním kódu, lze v tomto programu udělat během pár minut.

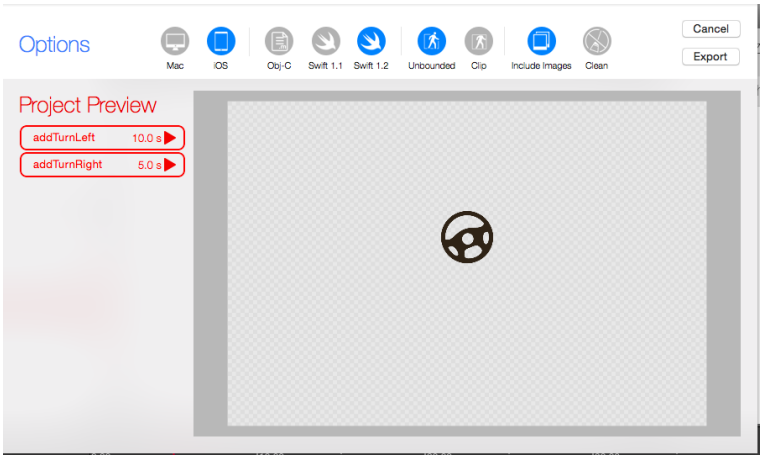
Core Animator má jednoduché moderní grafické rozhraní, jak můžete vidět na obrázku. Práce v Core Animatoru funguje následovně, do projektu si vložíte nějaký obrázek, kterému přidáte animace. Animace jsou následně vidět v levém sloupci v kolonce animation, kde se momentálně edituje označená položka. V pravém sloupci je panel základních animací, které se dají na sprite aplikovat. V dolním panelu je časová osa, na které určujeme jak dlouho animace potrvá, také se tam nastavují vlastnosti animace, jako je vytváření smyček pro danou animaci apod.



Obrázek 9.1: Core Animator

Veškeré animace vytvořené v Core Animatoru se dají vyexportovat a díky tomu získáte soubor s kódem, který je v jazyce Swift nebo Objective-C. Na obrázku je vidět, že máme dvě animace addTurnLeft a addTurnRight, který trvají 5 a 10 vteřin. Dále v horním panelu je na výběr pro jaký operační

systém exportovat a v jakém jazyce. Při exportu vznikne soubor s kódem v daném jazyce, který přetáhnete do projektu a na daný sprite pak následně zavoláte jednoduše „wheel.addTurnLeftAnimation()“.



Obrázek 9.2: Core Animator - export

Ačkoliv tento software vypadá pěkně, není zdarma[14]. Pro export animací musíte mít koupenou licenci, která činí 99.99$.

10 Realizovaná aplikace

Hru, kterou jsem realizoval v rámci své bakalářské práce, bylo bludiště pro dva hráče v rámci bezdrátové osobní sítě. V realizované hře bude k dispozici výběr vlastního jména, které bude viditelné pro ostatní zařízení. Ve hře budou hráči mezi sebou komunikovat přímo tj. komunikace bude typu peer-to-peer. Hra bude obsahovat dva sprity, které představují hráče. Ovládání hry budou jednoduchá gesta pomocí tahů prstem. Cílem těchto spritů bude projít bludištěm a získat točící se sprite jablíčka. V bludišti se nacházejí dvě brány a hráči budou muset získat klíč, který bránu odemkne.

Hru jsem realizoval v jazyce Swift za použití frameworků Multipeer Connectivity, UIKit, Sprite Kit a AVFoundation. Při volbě jazyka jsem se rozhodoval mezi Swiftem a Objective-C, ačkoliv je jazyk Objective-C více rozšířený mezi uživateli ve světě, rozhodl jsem se pro poměrně mladý jazyk Swift, protože je moderní a přehledný.

10.1 MPCHandler

Jako první se musel inicializovat framework Multipeer Connectivity. Proto jsem vytvořil třídu zvanou „MPCHandler“, která bude mít za úkol inicializovat potřebné parametry pro framework Multipeer Connectivity. Třídu MPCHandler jsem vytvářel jako nový soubor Cocoa Class, protože je zapotřebí vytvořit třídu, která bude podtřídou třídy NSObject, a také bude zapotřebí na začátku souboru importovat framework Multipeer Connectivity. A poté jsem vytvořil funkci `setupMPC()`, která inicializuje potřebné parametry frameworku.

Listing 10.1: Inicializace framworku Multipeer Connectivity

```
setupMPC(){
 //inicializace nazvu zarizeni
 if name == ''''{
  peer = MCPeerID(displayName:UIDevice.currentDevice().name)
 }else{
  peer = MCPeerID(displayName: name)
 }

 //inicializace sessionu
 session = MCSession(peer: peer)
 session.delegate = self

 //nastaveni browseru
 browser = MCNearbyServiceBrowser(peer: peer,
  discoveryInfo: nil, serviceType = ''bakalarka'')
 browser.delegate = self

 //inicializace advertieru
 advertiser = MCNearbyServiceAdvertiser(peer: peer,
  discoveryInfo: nil, serviceType: ''bakalarka'')
}
```

Jak bylo zmíněno už v kapitole 8, jsou zapotřebí čtyři hlavní věci: MCPeerID, MCSession, MCNearbyServiceBrowser a MCNearbyServiceAdvertiser a následně je delegovat (aby je ostatní třídy viděly a mohly využívat jejich funkce). Jméno zařízení, které se bude ostatním zařízením zobrazovat, neboli MCPeerID si může uživatel sám zvolit, ale pokud nic nezvolí (tj. nic nezadá), tak se MCPeerID nastaví na jméno současného zařízení. U browseru a advertiseru je discoveryInfo. DiscoveryInfo slouží k tomu, aby se při nalezení ostatních zařízení odeslal objekt nebo řetezec, který ověří správnost služby.

Dále je vhodné vytvořit protokol, který bude obsahovat potřebné funkce, které uvidí ostatní třídy v projektu.

Listing 10.2: protocol MPCManagerDelegate

```
protocol MPCManagerDelegate{
 //funkce pro zpracovani nalezenych zarizeni v okoli
 func foundPeer()
 //kdyz nalezeny peer zmizi nebo se ztrati v dosahu
 func lostPeer()
 //kdyz prijde pozvani k pripojeni od peera
 func invitationWasReceived(fromPeer: String)
 //pripojeni s peerem
 func connectedWithPeer(peerID: MCPeerID)
}
```

Momentálně tento protokol není viditelný pro ostatní třídy. Aby jej viděly, stačí do souboru AppDelegate.swift, který je defaultně generován při vytváření projektu, dopsat na začátek „var mpcHandler = MPCManager()“.

Dále si framework MultipeerConnectivity při implementaci vynucuje některé funkce.

Listing 10.3: Důležité funkce frameworku Multipeer Connectivity

```
//1
func browser(browser: MCNearbyServiceBrowser!,
 foundPeer peerID: MCPeerID!,
 withDiscoveryInfo info: [NSObject : AnyObject]!) {
  foundPeers.append(peerID)
  delegate?.foundPeer()
}
//2
func browser(browser: MCNearbyServiceBrowser!,
lostPeer peerID: MCPeerID!) {
 for (index, aPeer) in enumerate(foundPeers){
  if aPeer == peerID {
    foundPeers.removeAtIndex(index)
    break
   }
 }
delegate?.lostPeer()
 }
//3
func browser(browser: MCNearbyServiceBrowser!,
 didNotStartBrowsingForPeers error: NSError!) {
  println(error.localizedDescription)
}
//4
func advertiser(advertiser: MCNearbyServiceAdvertiser!,
 didReceiveInvitationFromPeer peerID: MCPeerID!,
  withContext context: NSData!,
  invitationHandler: ((Bool, MCSession!) -> Void)!) {
    self.invitationHandler = invitationHandler
    delegate?.invitationWasReceived(peerID.displayName)
}
//5
func advertiser(advertiser: MCNearbyServiceAdvertiser!,
 didNotStartAdvertisingPeer error: NSError!) {
 println(error.localizedDescription)
}
```

Listing 10.4: Důležité funkce frameworku Multipeer Connectivity

```
//6
 func session(session: MCSession!, peer peerID: MCPeerID!,
 didChangeState state: MCSessionState) {
 switch state{
  case MCSessionState.Connected:
 println("Pripojen ke sessionu: \(session)")
  delegate?.connectedWithPeer(peerID)
 case MCSessionState.Connecting:
  println("Pripojuji se k sessionu: \(session)")
 case MCSessionState.NotConnected:
   NSNotificationCenter.defaultCenter().
   postNotificationName("disconnected", object: peerID)
 default:
    println("Nepodarilo se pripojit k sessionu: \(session)")
   }
 }
//7
func session(session: MCSession!,
didReceiveData data: NSData!, fromPeer peerID: MCPeerID!) {
  let userInfo = ["data":data, "peerID":peerID]
 dispatch_async(dispatch_get_main_queue(), { () -> Void in
    NSNotificationCenter.defaultCenter().
    postNotificationName("MPC_DidReceiveDataNotification",
    object: nil, userInfo: userInfo)
 })
}
```

Popis funkcí u komentářů v kódech 10.3 a 10.4:

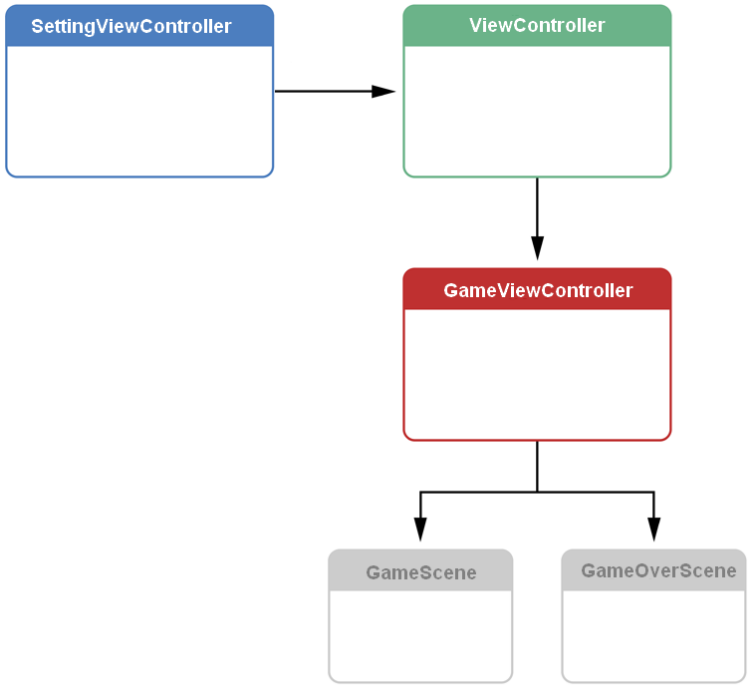
1. Při nalezení peera se přidá peer do UITableView, co je ve třídě View-Controller.swift, která je uvedena níže v sekci.
2. V případě, že peer zmizí, odejme se z UITableView
3. Pokud browser nezačal ještě vyhledávat zařízení v okolí, vypíše se chyba.
4. V případě, že jsme dostali pozvání od jiného uživatele, provede se daná funkce.
5. Výpis při selhání advertisera.

6. Session má 3 stavy, připojen, připojuje se a nepřipojen. Při změně jednotlivých stavů se provede daná akce.

7. Když dostaneme od uživatele data, vytvoříme slovník, který bude obsahovat data a od koho přišla a pošleme data pomocí notifikací, kde si ho převezme notifikace, která ho obsluhuje.

10.2 Obrazovky

Aplikaci obsluhují tři hlavní obrazovky 10.1: SettingViewController, ViewController a GameViewController.

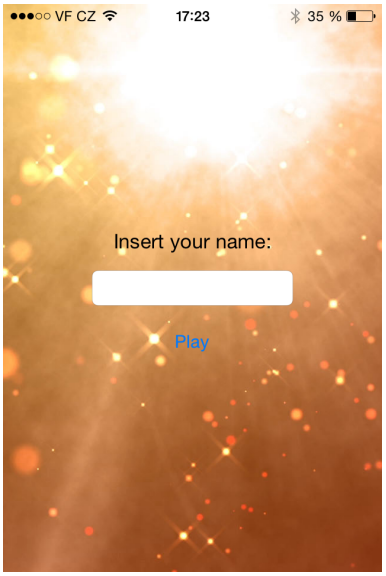


Obrázek 10.1: Propojení obrazovek v ukázkové hře

10.2.1 SettingViewController

SettingViewController je úvodní obrazovka, která se objeví při spuštění aplikace. V kódu této obrazovky je implementovaný UIKit framework a na

obrazovce jsou tři elementy viz obrázek 10.2.



Obrázek 10.2: SettingViewController

V této obrazovce má uživatel možnost zvolit si své peer jméno, které se bude zobrazovat ostatním uživatelům, proto je zapotřebí nějak vhodně nastavit UITextField. Ke kódové úpravě UITextFieldu je nejdřív zapotřebí dědit třídu UITextFieldDelegate, poté aplikovat jednotlivé vlastnosti, které chceme, například maximální počet znaků, schovat klávesnici apod.

Listing 10.5: Úprava UItextField

```
class SettingViewController: ... ,UITextFieldDelegate{
  var peerName: UITextField?
          .
          .
          .
 //skryt klavesnici
 func textFieldShouldReturn(textField:UItextField) -> Bool{
  testField.resignFirstResponder()
  return true
 }
 //omezeni textoveho pole na 15 libovolnych znaku
 func textField(textField: UItextField,
  shouldChangeCharactersInRange range: NSRange,
  replacementString string:String) -> Bool{
   let newLength = count(textField.text) + count(string)
   - range.length
   return newLength <=15
  }
}
```

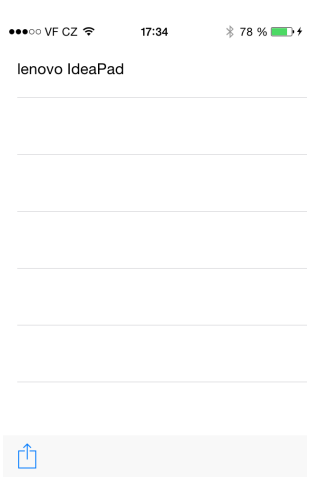
Posledním úkolem této třídy je zavolat funkci setupMPC() ze třídy MPCHandler. Funkce se zavolá až když uživatel stiskne tlačítko „Play“ a zároveň se přepne do obrazovky s id „toList“ tj. ViewController, kde se budou zobrazovat ostatní hráči.

Listing 10.6: Přesun do ViewControlleru

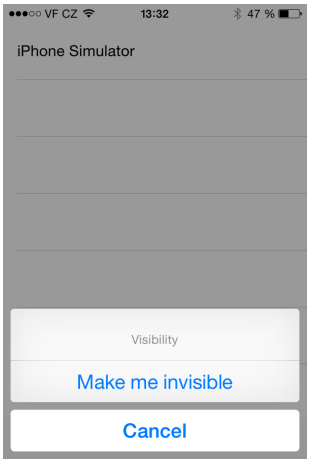
```
class SettingViewController: ... {
 let appDelegate =
 UIApplication.sharedApplication().delegate as! AppDelegate
        .
        .
        .
  @IBAction func playButton(sender: AnyObject) {
    if(peerName?.text != nil){
        appDelegate.mpcManager.name = peerName?.text
    }
    appDelegate.mpcManager.setupMPC()

    NSOperationQueue.mainQueue().addOperationWithBlock{
        () -> Void in
    self.performSegueWithIdentifier("toList", sender: self)
   }
}
```

10.2.2 ViewController

ViewController je třída, která ma starost zobrazení jednotlivých zařízení v okolí. Jednotlivá jména zařízení se zobrazují v UITableView 10.3. Kromě zobrazování má také na starost vysílání pomocí advertiser a hledání pomocí browser. U advertiseru je přidaná možnost vypnout advertiser, aby vás ostatní zařízení neviděla 10.4.



Obrázek 10.3: Seznam hráčů



Obrázek 10.4: Nastavení viditelnosti

Listing 10.7: ViewController

```
class ViewController: ..., MPCManagerDelegate{
 let appDelegate =
 UIApplication.sharedApplication().delegate as! AppDelegate
            .
            .
            .
 override func viewDidLoad() {
            .
            .
            .
   //zacit vyhledavat zarizeni v okoli
   appDelegate.mpcManager.browser.startBrowsingForPeers()
   //zacit vysilat ostatnim zarizenim
   appDelegate.mpcManager.advertiser.startAdvertisingPeer()

    }

  @IBAction func startStopAdvertising(sender: AnyObject) {
   let visibilityAction: UIAlertAction =
 UIAlertAction( ... ) { (alertAction) -> Void in
   if self.isAdvertising == true {
  //vypnout vysilani
appDelegate.mpcManager.advertiser.stopAdvertisingPeer()
  }else{
 //zapnout vysilani
appDelegate.mpcManager.advertiser.startAdvertisingPeer()
  }
 }
}
```

Při stiknutí na položku s jménem uživatele v UITableView, se uživateli odešle pozvání.

Listing 10.8: Poslání pozvánky

```
//1
func tableView(tableView: UITableView,
didSelectRowAtIndexPath indexPath: NSIndexPath) {
let selectedPeer =
appDelegate.mpcManager.foundPeers[indexPath.row] as MCPeerID
appDelegate.mpcManager.browser.invitePeer(selectedPeer,
toSession: appDelegate.mpcManager.session, withContext: nil,
timeout: 20)
}
//2
func invitationWasReceived(fromPeer: String) {
 //hrac s vami chce hrat hru//
 //prijmout//
 self.appDelegate.mpcManager.
invitationHandler(true, self.appDelegate.mpcManager.session)
 }
//odmitnout//
self.appDelegate.mpcManager.invitationHandler(false, nil)
}
   ...
}
//3
func connectedWithPeer(peerID: MCPeerID) {
NSOperationQueue.mainQueue().addOperationWithBlock {
 () -> Void in
self.performSegueWithIdentifier("toGame", sender: self)
 }
}
```

Popis funkcí u komentářů v kódu 10.8:

1. Při výběru položky v UITablevView se dané osobě pošle pozvánka ke hře.
2. Pokud jsme byli pozvaní, zobrazí se hláška a můžete pozvání přijmout či odmítnout.
3. Pokud uživatel přijal výzvu, přepneme se do okna hry.

10.3 GameViewController

GameViewController je obrazovka, která ma na starost dvě scény. První scénou je samotná hra (GameScene.swift) a druhou scénou je scéna s oznámením výhry (GameOverScene.swift). Soubory GameViewController.swift a GameScene.swift se vytváří defaultně při vytváření projektu a obsahují už nějaké defaultní inicializace, které většinou můžeme kompletně smazat. Ale co je důležité udělat jako první, je v souboru GameViewController.swift incializovat scénu GameScene.swift, kde bude kód naší hry.

Listing 10.9: Inicializace scény

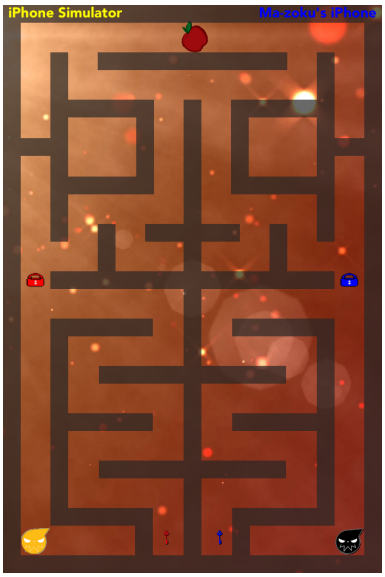
```
override func viewDidLayoutSubviews() {
 let skView = self.view as! SKView
 if skView.scene == nil {
 //zobrazeni fps
   skView.showsFPS = true
//zobrazeni poctu nodu
   skView.showsNodeCount = true
 let gameScene = GameScene(size: skView.bounds.size)
 gameScene.scaleMode = SKSceneScaleMode.AspectFill

 skView.presentScene(gameScene)
 }
}
```

10.4 GameScene

10.4.1 Vykreslení hry

K simulaci herního světa je zapotřebí velké množství kalkulací a k tomu máme OpenGL ES, který je navržen pro mobilní zařízení. OpenGL je obtížné se naučit, ale Sprite Kit má v sobě vestavěn OpenGL ES tak, že můžeme jednoduše přistupovat ke všemu, co potřebujeme pro tvorbu 2D her, aniž bychom si museli dělat starosti s OpenGL.

Při tvorbě 2D her se většinou využívají obrázky (nazývané sprite), které na obrazovce představují hráče, překážky apod. Na obrázku 10.5 je vidět,



Obrázek 10.5: Ukázka hry

co všechno představují v realizované hře sprity (černý hráč, žlutý hráč, zdi, brány, klíče a jablíčko).

10.4.2 Přidávání spritů

Ve Sprite Kitu je přidávání spritů velice jednoduché, stačí určit obrázek spritu a pozici kam jej přidat.

Listing 10.10: Přidání spritů

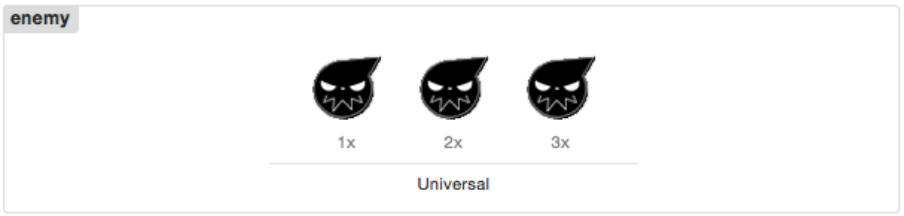
```
let enemy = SKSpriteNode(imageNamed: "enemy")
enemy.position = CGPointMake(200.0, 150.0)
self.addChild(enemy)
```

CGPoint je datová strukura pro geometrické účely, je reprezentována jako dvourozměrný bod. Také je důležité neopomenout to, že středový bod spritu je ve středu obrázku, pokud se nenastaví jinam.

10.4.3 Fyzikální svět

SKScene ve Sprite Kitu mají defaultně nastaveny základní vlastnosti fyzického světa. Jedním z nich je gravitace.Na každý vytvořený sprite se aplikuje vliv gravitace s vektorovým směrem dolů, pokud ji nevypnete. V mé ukázkové hře gravitace nebyla zapotřebí, takže jsem nastavil gravitaci na hodnotu nula, tudíž nulová gravitace.

10.4.4 Vykreslení scény s image sety



Obrázek 10.6: Sada obrázku

Při vývoji her pro iOS se většinou přidávají tři sady jednoho obrázku s různými velikostmi označené jako @1x, @2x, a @3x. Je to kvůli tomu, že zařízení od Apple mají různě veliké obrazovky. @1x je označení pro starší generace zařízení (např. iPhone 3, 3S). @2x je pro zařízení iPhone 4, 4S, 5 a 5S, a @3x je pro iPhone 6 a 6 plus. Podle těchto označení si Sprite Kit vybírá obrázky v závislosti na tom, na jakém zařízení je.

Výběr spritů o různých velikostech by mohl být v mnou realizované aplikaci problém. Obrazovka u iPhone 6 má mnohem víc bodů než u iPhonu 4S a to by mohlo zapříčinit nerovnováhu hry, protože pro hráče s větší obrazovkou (tedy např. iPhone 6) bude trvat větší dobu ujít z jednoho rohu do druhého než u hráče s menší obrazovkou. Pro vyřešení tohoto problému jsem musel nastavit scénu na fixní velikost v bodech a nastavit scaleMode viz. 10.9 na aspectFill. To zapříčiní to, že oba hráči budou mít stejnou velikost scény, ale obrázky se roztáhnou nebo zúží podle šířky u obrazovky zařízení a zachová poměr mezi výškou a šířkou.

10.4.5 Kolize a doteky

Ve Sprite Kitu se dá nastavit každému spritu fyzické tělo. Fyzické tělo sprita může mít tvar kruhu, obdélníku či čtyřúhelníku. Teď když mají sprity fyzické tělo, tak samozřejmě mohou do sebe narážet, a nejen to, také pro ně platí dynamika hmotného bodu. Když dva sprity do sebe narazí, tak se od sebe odrazí silou, která zavisí i na tom, jakou hustotu těla má daný sprite nastavenou, apod.

Někdy je zapotřebí, aby jeden sprite nekolidoval se všemy sprity, ale jen některými typy spritů, proto se u spritů musí nastavit bitmasky. V mém hře viz obrázek 10.5 je hráč, který koliduje pouze se zdí a bránou s druhým hráčem a s jablkem nekoliduje, proto jsem je musel rozřadit do různých bitmask.

Bitmasky, které jsem v aplikaci využil byly datové typy 32b integerů. Proč 32b integery? Protože budeme mít k dispozici 32 bitů, což znamená, že můžeme mít až 32 různých spritů s tím, že vždycky s každým novým spritem budeme posouvat jedničku o jednu vlevo.

Listing 10.11: Bitmasky

```
//1
let hrac1Category:UInt32 = 0x1 << 0
let hrac2Category:UInt32 = 0x1 << 1
let jablkoCategory:UInt32 = 0x1 << 2
let zedCategory:UInt32 = 0x1 << 3
//2
hrac1.physicsBody =
SKPhysicsBody(circleOfRadius: hrac1.frame.size.width/2)
hrac2.physicsBody =
SKPhysicsBody(circleOfRadius: hrac2.frame.size.width/2)
zed.physicsBody =
SKPhysicsBody(rectangleOfSize: zed.frame.size)
//3
hrac1.physicsBody?.categoryBitMask = hrac1Category
hrac2.physicsBody?.categoryBitMask = hrac2Category
zed.physicsBody?.categoryBitMask = zedCategory
```

Popis funkcí u komentářů v kódu 10.11:

1. Vytvoření jednotlivých kategorií pro jednotlivé sprity.

2. Přidání spritům jejich fyzická těla, pro hráče jsou fyzická těla kruh o jejich vlastním poloměru. Pro zeď je to obdélník o vlastní velikosti.

3. Přiřazení bitmask jednotlivým spritům.

Nyní mají sprity určené bitmasky jejich fyzických těl, tak je vhodné nastavit kdo s kým koliduje a kdo se s kým dotýká. V mé hře hráč koliduje pouze se zdí a s bránou, dotýká se jablka, klíče a brány.

Listing 10.12: Kolize

```
//doteky
hrac1.physicsBody?.contactTestBitMask =
jablkoCategory | klicCategory | branaCategory
//kolize
hrac.physicsBody?.collisionBitMask =
zedCategory | branaCategory
```

10.4.6 Detekce dotyku

Sprite Kit má funkci, která detekuje kolize spritů, v mé hře hráč má nastavený dotek s klíčem, jablkem a bránou. V případě, že se dotkne klíče, vezme si k sobě klíč a spustí se zvuk cinknutí klíče.

Listing 10.13: Přidání zvuku

```
//1
let backGroundMusicPlayer = AVAudioPlayer()
let bgMusicURL = NSBundle.mainBundle().
 URLForResource("keySound", withExtension: "mp3")
backGroundMusicPlayer = AVAudioPlayer(
 contentsOfURL: bgMusicURL, error: nil)
backGroundMusicPlayer.prepareToPlay()
//2
func didBeginContact(contact: SKPhysicsContact) {
 var firstBody = SKPhysicsBody()
 var secondBody = SKPhysicsBody()
if contact.bodyA.categoryBitMask<contact.bodyB.
categoryBitMask{
 firstBody = contact.bodyA
 secondBody = contact.bodyB
}else{
firstBody = contact.bodyB
 secondBody = contact.bodyA
 }
//3
if firstBody.categoryBitMask == hrac1Category
&& secondBody.categoryBitMask == klicCategory{
hrac1MaKey = true
secondBody.node?.removeFromParent()
backGroundMusicPlayer.play()
}
```

- 1. Inicializace přehrávače pro přehrávání hudby
- 2. Při detekci srážky dvou spritů, seřadíme si sprity podle velikosti bitmasky a vložíme do proměných.
- 3. Kategorie hráče je vždycky menší než ostatní, proto můžeme v klidu říct, že první tělo je hráč a druhé tělo je klíč. Pokud tvrzení souhlasí, vezme si hráč klíč a spustí se zvuk.

10.4.7 Ovládání hry

Pro hry typu bludiště je dle mého názoru nejlepší ovládání typu táhnutí prstem na stranu, na kterou chceme. Nejdřív se musí vytvořit kolekce směrů a poté inicilizovat rozpoznávání tahů prstem.

Listing 10.14: Kolekce směrů

```
//smery
 enum DirectionMove: String{
         case East = "right"
         case West = "left"
         case North = "up"
         case South = "down"
         case Default = "none"
 }
//tah doprava
let swipeRight:UISwipeGestureRecognizer =
UISwipeGestureRecognizer(target: self,
action: Selector("swipedRight:"))
swipeRight.direction = .Right
view.addGestureRecognizer(swipeRight)
```

Při detekci tahu doprava se zavolá funkce „swipedRight“ a podle toho se figurka hráče začne pohybovat doprava a musí také oznámit soupeře o pohybu. Při tvorbě peer-to-peer hry, se musíme na hru dívat jako na jednu hru, kterou obsluhuje několik vláken (v tomto případě notifikace) a na základě toho provádět danou obsluhu. Při tahu doprava se informace o změně pohybu se pošle druhému hráči jako slovník ve formátu JSON.

Listing 10.15: Odesílání dat

```
func swipedRight(sender: UISwipeGestureRecognizer){
hrac1.direction = .East

let messageDict = ["direction":"right"]
let messageDataObject =
NSJSONSerialization.dataWithJSONObject(messageDict,
 options: NSJSONWritingOptions.PrettyPrinted, error: nil)
var error: NSError?

 appDelegate.mpcManager.session.sendData(messageDataObject,
  toPeers: appDelegate.mpcManager.session.connectedPeers,
  withMode: MCSessionSendDataMode.Reliable, error: &error)
}
```

Jak je vidět v kódu 10.15, tak při obdržení dat od druhého hráče se data pošlou do notifikace s názvem „MPC_DidReceiveDataNotification“. A v naší scéně hry máme observera, který čeká, až mu někdo oznámí příjem dat.

Listing 10.16: Notifikace

```
func swipedRight(sender: UISwipeGestureRecognizer){
NSNotificationCenter.defaultCenter().addObserver(self,
 selector:"receiverHandler:",
 name:"MPC_DidReceiveDataNotification",object: nil)
```

Jakmile observer získá oznámení o příjmu dat, zavolá funkci, která data přijme, rozbalí a zpracuje.

Listing 10.17: Zpracování příchozích dat

```
func receiverHandler(notification:NSNotification){

let userInfo = notification.userInfo! as Dictionary
let receiveData:NSData = userInfo["data"] as! NSData

let msg = NSJSONSerialization.JSONObjectWithData(
receiveData, options: NSJSONReadingOptions.AllowFragments,
error: nil) as! NSDictionary

let senderPeerID:MCPeerID = userInfo["peerID"] as! MCPeerID
let senderDisplayName = senderPeerID.displayName

if msg.objectForKey("direction")?.isEqualToString("right"){
 hrac2.direction = .East
 }
}
```

Pohyb figurky hráče je zprostředkován ve funkci update(), která se mění v rychlosti snímků. Hráč se pohybuje konstantní rychlostí, kterou nastavíme.

Listing 10.18: Pohyb spritu

```
override func update(currentTime: NSTimeInterval) {
if direction == ''right''{
  hrac.position.x += rychlost
 }
}
```

Je ovšem dobré vědět, že Multipeer Connectivity nefunguje na pozadí. To znamená, že při minimalizaci aplikace je Multipeer Connectivity neaktivní tudíž spojení bude přerušeno, a funkci reconnect (opětovné připojení) neumí.

10.5 Testování aplikace

Aplikaci jsem testoval na svém fyzickém zařízení (iPhone 4S), které je zaregistrováno v univerzitním iOS University Developer programu, a v dostupných simulátorech v XCodu.

- **Ovládání hry:** Ovládaní hry fungovalo jak na fyzickém zařízení, tak i na simulátoru podle předpokladů a bezchybně. Detekce a reakce tahů prstem na obou zařízeních byla okamžitá.
- **Funkčnost hry:** Na obou zařízeních hra fungovala podle očekávání. Sprity reagovali na jednotlivé situace správně tak jak mají a bez větších problémů.
- **Běh hry:** Na mém fyzickém zařízení (iPhone 4S) běžela hra svižně a bez problému se snímkovou frekvencí 60 snímků za sekundu. Což je správná rychlost, která je vhodná pro hraní her.

  Ovšem, u simulátorů byl běh aplikace trochu problém. Sprite Kit využívá Open GL ES pro zrychlení výkresu na fyzických zařízeních. Ale iOS simulátor simuluje OpenGL ES přes softwarový renderer, který je mnohem pomalejší bez ohledu na to, jak výkonný je procesor na vašem počítači, kde vyvíjíte aplikaci. Běh aplikace se mi lišil na různých typech simulátoru, což zkreslovalo výsledky hry. Na simulátoru iPhone 5S byla snímková frekvence kolem 50 snímků za sekundu. Na simulátoru iPhonu 6 byla rychlost 20 snímků za sekundu. A na simulátoru iPadu Air byla snímková rychlost necelých 8 snímků za sekundu.

10.6 Instalace aplikace

Instalace aplikací u platformy iOS je pouze možná přes internetový obchod App Store. Digitální distribuce aplikací na App Store je umožněno pouze lidem, co mají placený vývojářský účet. Vývoj mé aplikace probíhal v rámci univerzitního vývojářského programu, který společnost Apple Inc. poskytuje univerzitám zdarma. V rámci tohoto programu není možná distribuce aplikace na App Store tudíž pro běžného uživatele není instalace této aplikace možná.

Jediná možnost instalace aplikace je mít fyzické zařízení, které je zaregistrováno a spojeno s účtem, který je ve vývojářském programu společnosti Apple Inc. a následně aplikaci nainstalovat přes XCode.

11 Závěr

Cílem bakalářské práce bylo prozkoumat možnosti vývoje multiuživatelských her pro platformu iOS a vybranné frameworky pro tvorbu her. Popsané frameworky v této práci nejsou omezeny jen na tvorbu her, ale i komplexnější aplikace.

Na základě těchto znalostí jsem navrhl a realizoval multiuživatelskou hru typu bludiště. Hra je pro dva hráče s komunikací typu peer-to-peer. V rámci této práce se může čtenář seznámit se s architekturou operačního systému iOS a technologiemi, které ovladají jednotlivé vrsty tohoto operačního systému. Dále jsou v této práci krátce popsány základy programovacími jazyky Objective-C a Swift, které slouží jako primární jazyk pro vývoj softwaru na platformách OS X a iOS.

Při tvorbě aplikace byly použity pouze volně dostupné frameworky a knihovny. Přestože je Objective-C stále dominujicím jazykem pro vývoj aplikací pro iOS, byla aplikace napsaná v poměrně mladém jazyce Swift. Aplikace je vytvořena za pomocí frameworků Sprite Kit, UIKit , AVFoundation a Multipeer Connectivity.

Vzniklá hra představuje příklad práce se Sprite Kitem a propojení mezi scénou hry a okny aplikace. Dále ukazuje způsob implementace Multipeer Connectivity pro komunikaci mezi zařízeními. Práce ukazuje jeden z možných postupů při vývoji her pro platformu iOS.

Literatura

[1] *About the iOS Technologies.*[on-line][2014-09-17]
https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[2] *Sprite Kit.*[on-line][2014-09-17]
https://developer.apple.com/library/mac/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html

[3] *UIKit Framework Reference.*[on-line][2014-09-17]
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/

[4] *The Foundation Framework.*[on-line][2013-10-22]
https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/ObjC_classic/index.html

[5] *AVFoundation.*[on-line][2015-03-29]
https://developer.apple.com/av-foundation/

[6] *Multipeer Connectivity.*[on-line][2013-09-18]
https://developer.apple.com/library/prerelease/ios/documentation/MultipeerConnectivity/Reference/MultipeerConnectivityFramework/index.html

[7] William Herring. *Learn iOS Game Development By Example: 10 Projects to Get You Started.*[on-line][2011-06-20]
http://code.tutsplus.com/tutorials/

[8] Blow, J. *Game Development: Harder Than You Think,* 2004. Queue, Vol. 1(10), ss. 2837, February 2004.

[9] *SkipCast.*[on-line][2014-09-14]
`https://skipcasts.com/`.

[10] Wenderlich, R. *Tutorial for developers & gamers.*[on-line][2014-08-21]
`http://www.raywenderlich.com`

[11] *XCode.*[on-line][2014-05-02]
`http://en.wikipedia.org/wiki/Xcode`

[12] *Swift (programming language).*[on-line][2014-05-01]
`http://en.wikipedia.org/wiki/Swift_%28programming_language%29`

[13] *The LLVM Compiler Infrastructure.*[on-line][2014-04-07]
`http://llvm.org/`

[14] *CoreAnimator.*[on-line][2015-02-25]
`http://www.coreanimator.com/`

[15] *iOS Frameworks.*[on-line][2014-09-17]
`https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSFrameworks/iPhoneOSFrameworks.html`

[16] David Mark, Jack Nutting, Kim Topley , Fredrik Olsson, Jeff LaMarche *Beginning iPhone Development with Swift: Exploring the iOS SDK,* 2014. ISBN: 978-1484204108

[17] Boisy G. Pitre *Swift for Beginners: Develop and Design,* 2014. ISBN: 978-0134044705

Seznam zkratek

- **API:** Application Programming Interface, rozhraní, které slouží jako knihovna, kterou programátor využívá.
- **GCC:** GNU Compiler Collection, sada překladačů pro GNU.
- **GNU:** Svobodný operační systém unixového typu.
- **IDE:** Integrated Development Environment, vývojové prostředí.
- **iOS:** iPhone Operating System, operační systém pro zařízení iPhone.
- **LLVM:** Low-Level Virtual Machine.
- **MVC:** Mode-View-Controller, návrhový vzor model-pohled-řadič.
- **NeXT:** Generace mikropočítačů.
- **NeXTSTEP:** Objektově orientovaný, multitaskingový operační systém.
- **NeXT/Sun:** Objektově orientovaný, multitaskingový operační systém.
- **OpenGL:** Open Graphic Library, multiplatformí rozhraní pro tvorbu počítačové grafiky.
- **OpenGL ES:** Open Graphic Library for Embedded Systems, OpenGL pro vykreslování 2D a 3D grafiky.
- **SDK:** Software Development Kit nástroj pro vývoj sowftwaru.
- **UI:** User Interface, uživatelské rozhraní.
- **XML:** Extensible Markup Language, značkovací jazyk.

Příloha A

Instalační dokumentace

Požadavky

K instalaci aplikace na fyzické zařízení je zapotřebí XCode verze 6.3 s verzí Swiftu 1.2. Dále je nutné mít fyzické zařízení s operačním systémem iOS 8.0 a vyšší.

Také je zapotřebí mít *team provisioning profile*, kde zaregistrujete své fyzické zařízení, kterému budou následně poskytnuty potřebné certifikáty pro instalaci aplikace.

Překlad a instalace programu

Program přeložíte stisknutím tlačítka „build“ v XCodu a následně je nainstalujete na zařízení výběrem vašeho zařízení a stiknutí tlačítka „run“.

Příloha B

Uživatelská dokumentace

Spuštění aplikace

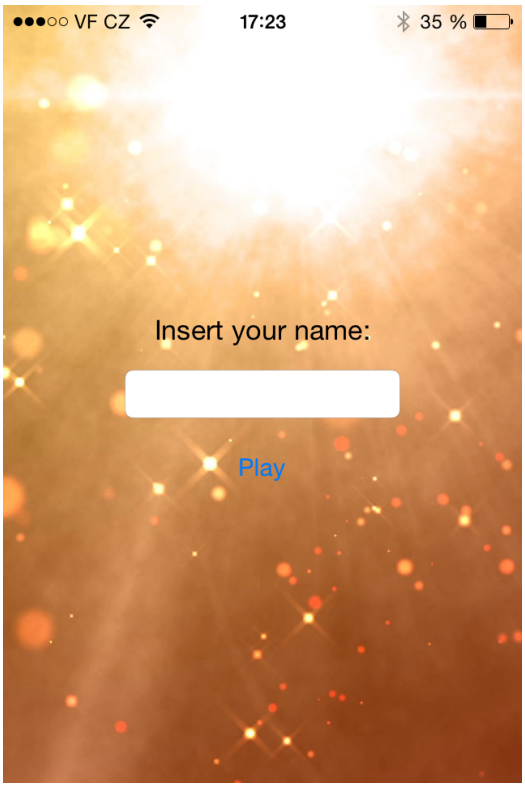
Aplikaci spustíte pokliknutím na aplikaci na vašem fyzickém zařízení.

Ovladání aplikace

Po spuštění aplikace se zobrazí hlavní okno, kde si zvolíte své jméno (max 15 libovolných znaků) a stiknete tlačítko „play“ viz 11.1, který vás přesměruje na seznam hráčů.

Pokliknutím na jméno soupeře v seznamu hráčů, pošlete soupeři pozvání ke hře. Pokud soupeř pozvání přijme, přejdete do hry v opačném případě se nic nestane 11.2. V dolním ovládacím panelu pod seznamem hráčů je na výběr možnosti zneviditelnit se či zviditelnit ostatním hráčům.

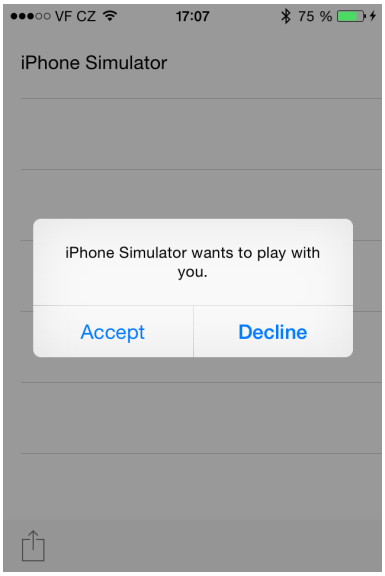
V případě, že soupeř přijme pozvání ke hře, přepnete se do hry viz 11.3. Hru ovládáte pomocí tahů prstem do čtyř stran (doprava, doleva, nahoru a dolů). Bránu otevřete pomocí klíčů. Pokud seberete klíč, objeví se vedle vašeho jména, a to znamená, že vlastníte daný klíč. Pokud vlastníte klíč a dotknete se brány, brána se otevře a váš klíč zmizí.



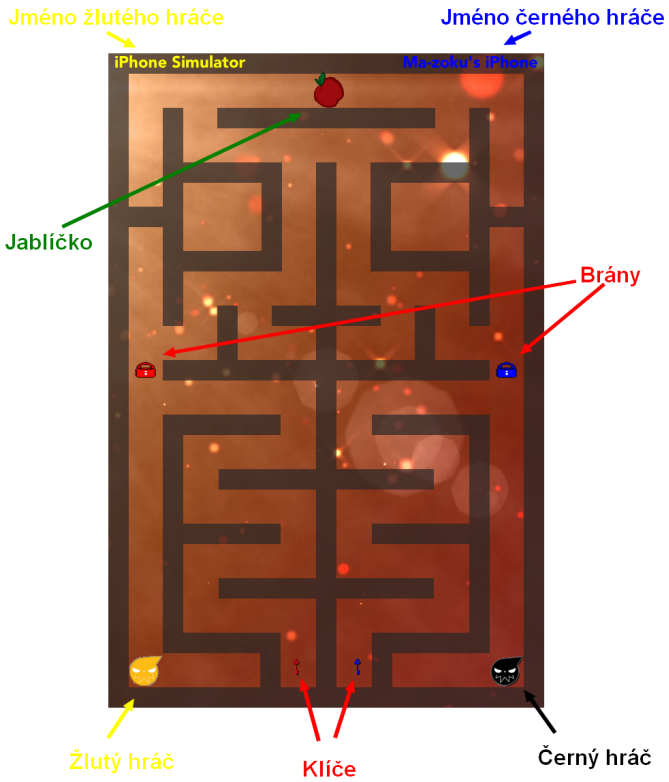
Obrázek 11.1: Hlavni okno hry

Cíl hry

Cílem hry je měnit směr své figurky pomocí tahu prstem, tak abyste získali jablíčko dřív než soupeř. Vyhrává ten, kdo sebere jablíčko jako první.



Obrázek 11.2: Pozvání od hráče



Obrázek 11.3: Hra